CS 61C: Great Ideas in Computer Architecture (Machine Structures) Caches Part 3

Instructors:
Krste Asanović & Randy H. Katz
http://inst.eecs.berkeley.edu/~cs61c/
• Parallel Requests
  Assigned to computer
e.g., Search “Katz”

• Parallel Threads
  Assigned to core
e.g., Lookup, Ads

• Parallel Instructions
  >1 instruction @ one time
e.g., 5 pipelined instructions

• Parallel Data
  >1 data item @ one time
e.g., Add of 4 pairs of words

• Hardware descriptions
  All gates @ one time

• Programming Languages

You Are Here!

Today’s Lecture

Software

hardware

Harness
Parallelism &
Achieve High
Performance

Software

Hardware

Warehouse
Scale
Computer

Smart
Phone

Computer

Core

Memory
(Cache)

Input/Output

Main Memory

Instruction Unit(s)

Functional
Unit(s)

Logic Gates

Today’s Lecture

10/19/17

Fall 2017 - Lecture #16
Typical Memory Hierarchy

- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the cheapest technology at the ≈ speed offered by the fastest technology.
Review: Direct-Mapped Cache

- One word blocks, cache size = 1K words (or 4KB)

Valid bit ensures something useful in cache for this index

Compare Tag with upper part of Address to see if a Hit

Read data from cache instead of memory if a Hit
Review: Four-Way Set-Associative Cache

$2^8 = 256$ sets each with four ways (each with one word block)

# Sets x # Ways = # Blocks
# Blocks x Bytes/Block = Cache Capacity
Review: Range of Set-Associative Caches

• For a *fixed-size* cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Word offset</th>
<th>Byte offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used for tag compare</td>
<td>Selects the set</td>
<td>Selects the word in the block</td>
<td></td>
</tr>
</tbody>
</table>

- Decreasing associativity, less ways, more sets
  - Direct mapped (only one way)
  - Smaller tags, only a single comparator

- Increasing associativity, more ways, less sets
  - Fully associative (only one set)
  - Tag is all the bits except word and byte offset

# Sets x # Ways = # Blocks
# Blocks x Bytes/Block = Cache Capacity
Review: Costs of Set-Associative Caches

• N-way set-associative cache costs
  – N comparators (delay and area)
  – MUX delay (set selection) before data is available
  – Data available after set selection (and Hit/Miss decision).
    DM &: block is available before the Hit/Miss decision
      • In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss
• When miss occurs, which way’s block selected for replacement?
  – Least Recently Used (LRU): one that has been unused the longest (principle of temporal locality)
    • Must track when each way’s block was used relative to other blocks in the set
    • For 2-way SA $, one bit per set → set to 1 when a block is referenced; reset the other way’s bit (i.e., “last used”)
Review: Cache Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache block for eviction

- Least-Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement

- Example of a Simple “Pseudo” LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whenever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
  - (example of “not-most-recently used” replacement policy)
Review: Write Policy Choices

• Cache Hit:
  – **Write through**: writes both cache & memory on every access
    • Generally higher memory traffic but simpler pipeline & cache design
  – **Write back**: writes cache only, memory written only when dirty entry evicted
    • A dirty bit per line reduces write-back traffic
    • Must handle 0, 1, or 2 accesses to memory for each load/store

• Cache Miss:
  – **No write allocate**: only write to main memory
  – **Write allocate** (aka fetch on write): fetch into cache

• Common combinations:
  – Write through and no write allocate
  – Write back with write allocate
Review: Benefits of Set-Associative Caches

- Choice of DM $vs.$ SA $depends on cost of a miss vs. cost of implementation

- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)
Review: Average Memory Access Time (AMAT)

• Average Memory Access Time (AMAT) is the average time to access memory considering both hits and misses in the cache

\[
AMAT = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}
\]
Outline

• Understanding Cache Misses
• Increasing Cache Performance
• Performance of multi-level Caches (L1, L2, ...)
• Real world example caches
• And in Conclusion ...
Outline

- Understanding Cache Misses
  - Increasing Cache Performance
  - Performance of multi-level Caches (L1, L2, ...)
  - Real world example caches
  - And in Conclusion ...
Miss Rate vs. Cache Size on the Integer Portion of SPECCPU2000
Sources of Cache Misses (3 C’s)

• **Compulsory** (cold start, first reference):
  – 1st access to a block, not a lot you can do about it
    • If running billions of instructions, compulsory misses are insignificant

• **Capacity**:
  – Cache cannot contain all blocks accessed by the program
    • Misses that would not occur with infinite cache

• **Conflict** (collision):
  – Multiple memory locations mapped to same cache set
    • Misses that would not occur with ideal fully associative cache
How to Calculate 3C’s Using Cache Simulator

1. **Compulsory**: set cache size to infinity and fully associative, and count number of misses

2. **Capacity**: Change cache size from infinity, usually in powers of 2, and count misses for each reduction in size
   - 16 MB, 8 MB, 4 MB, ... 128 KB, 64 KB, 16 KB

3. **Conflict**: Change from fully associative to n-way set associative while counting misses
   - Fully associative, 16-way, 8-way, 4-way, 2-way, 1-way
• Three sources of misses (SPEC2000 integer and floating-point benchmarks)
  – Compulsory misses 0.006%; not visible
  – Capacity misses, function of cache size
  – Conflict portion depends on associativity and cache size
Outline

• Understanding Cache Misses
• Increasing Cache Performance
• Performance of multi-level Caches (L1, L2, ...)
• Real world example caches
• And in Conclusion ...
Recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline

Must pipeline instruction along with data, so control operates correctly in each stage
Improving Cache Performance

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

- Reduce the time to hit in the cache
  - E.g., Smaller cache
- Reduce the miss rate
  - E.g., Bigger cache
- Reduce the miss penalty
  - E.g., Use multiple cache levels
Cache Design Space

Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

• 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 (L-cache, D-cache)
  – Depends on technology / cost
• Simplicity often wins
Break!
Peer Instruction

• For a cache with constant total capacity, if we increase the number of ways by a factor of two, which statement is false:
  A: The number of sets is halved
  B: The tag width decreases
  C: The block size stays the same
  D: The set index decreases
Peer Instruction

• For a cache with constant total capacity, if we increase the number of ways by a factor of two, which statement is false:
  A: The number of sets is halved
  B: The tag width decreases
  C: The block size stays the same
  D: The set index decreases
Peer Instruction

• For S sets, N ways, B blocks, which statements hold?
  (i) The cache has B tags
  (ii) The cache needs N comparators
  (iii) B = N x S
  (iv) Size of Set Index = \( \log_2(S) \)

A : (i) only
B : (i) and (ii) only
C : (i), (ii), (iii) only
D : All four statements are true
Peer Instruction

- For $S$ sets, $N$ ways, $B$ blocks, which statements hold?
  - (i) The cache has $B$ tags
  - (ii) The cache needs $N$ comparators
  - (iii) $B = N \times S$
  - (iv) Size of Set Index = $\log_2(S)$

- **A**: (i) only
- **B**: (i) and (ii) only
- **C**: (i), (ii), (iii) only
- **D**: All four statements are true
Primary Cache Parameters

- **Block size**
  - How many bytes of data in each cache entry?

- **Associativity**
  - How many ways in each set?
  - Direct-mapped => Associativity = 1
  - Set-associative => 1 < Associativity < #Entries
  - Fully associative => Associativity = #Entries

- **Capacity (bytes)** = Total #Entries * Block size
- **#Entries** = #Sets * Associativity
Impact of Larger Cache on AMAT?

• 1) Reduces misses (what kind(s)?)
• 2) Longer Access time (Hit time): smaller is faster
  – Increase in hit time will likely add another stage to the pipeline
• At some point, increase in hit time for a larger cache may
  overcome improvement in hit rate, yielding a decrease in
  performance
• Computer architects expend considerable effort optimizing
  organization of cache hierarchy – big impact on performance
  and power!
Increasing Associativity?

• Hit time as associativity increases?
  – Increases, with large step from direct-mapped to >=2 ways, as now need to mux correct way to processor
  – Smaller increases in hit time for further increases in associativity

• Miss rate as associativity increases?
  – Goes down due to reduced conflict misses, but most gain is from 1->2->4-way with limited benefit from higher associativities

• Miss penalty as associativity increases?
  – Unchanged, replacement policy runs in parallel with fetching missing line from memory
Increasing #Entries?

• Hit time as #entries increases?
  – Increases, since reading tags and data from larger memory structures

• Miss rate as #entries increases?
  – Goes down due to reduced capacity and conflict misses
  – *Architects rule of thumb: miss rate drops ~2x for every ~4x increase in capacity (only a gross approximation)*

• Miss penalty as #entries increases?
  – Unchanged

  *At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance*
Increasing Block Size?

• Hit time as block size increases?
  – Hit time unchanged, but might be slight hit-time reduction as number of tags is reduced, so faster to access memory holding tags

• Miss rate as block size increases?
  – Goes down at first due to spatial locality, then increases due to increased conflict misses due to fewer blocks in cache

• Miss penalty as block size increases?
  – Rises with longer block size, but with fixed constant initial latency that is amortized over whole block
Administrivia

- Midterm #2 1.5 weeks away! October 31!
- Review Session: 10am-12pm on Saturday, Oct 28. Location: Hearst Annex A1
- Project 2-1 due Monday, Oct 23 at 11:59pm
  - We will release a project edX assignment in lab
  - Worth a small portion of project 2 points
  - Due on edX along with Project 2-1 (Monday night)
- Homework 3 due this Friday
- Reminder: notify your lab TA about a change in Project 2 partners
Peer Instruction

For a cache of fixed capacity and blocksize, what is the impact of increasing associativity on AMAT:

A: Increases hit time, decreases miss rate
B: Decreases hit time, decreases miss rate
C: Increases hit time, increases miss rate
D: Decreases hit time, increases miss rate
Peer Instruction

For a cache of fixed capacity and blocksize, what is the impact of increasing associativity on AMAT:

A: Increases hit time, decreases miss rate
B: Decreases hit time, decreases miss rate
C: Increases hit time, increases miss rate
D: Decreases hit time, increases miss rate
Peer Instruction

Impact of Larger Blocks on AMAT:

• For fixed total cache capacity and associativity, what is effect of larger blocks on each component of AMAT:
  
  A: Decrease
  B: Unchanged
  C: Increase

Hit Time?
Miss Rate?
Miss Penalty?
Peer Instruction

Impact of Larger Blocks on AMAT:

- For fixed total cache capacity and associativity, what is effect of larger blocks on each component of AMAT:
  
  A : Decrease
  B : Unchanged
  C : Increase

  - Hit Time?
  - Miss Rate?
  - Miss Penalty?

  Shorter tags +, mux at edge -
  C: Unchanged (but slight increase possible)
  A: Decrease (spatial locality; conflict???)
  C: Increase (longer time to load block)

  Write Allocation? It depends!
Impact of Larger Blocks on Misses:

- For fixed total cache capacity and associativity, what is effect of larger blocks on each component of miss:
  
  - A: Decrease
  - B: Unchanged
  - C: Increase

Compulsory?
Capacity?
Conflict?
Peer Instruction

Impact of Larger Blocks on Misses:

• For fixed total cache capacity and associativity, what is effect of larger blocks on each component of miss:

  A: Decrease
  B: Unchanged
  C: Increase

Compulsory? A: Decrease (if good Spatial Locality)
Capacity? B: Increase (smaller blocks fit better)
Conflict? A: Increase (more ways better!)
  Less effect for large caches
How to Reduce Miss Penalty?

• Could there be locality on misses from a cache?
  – Use multiple cache levels!
  – With Moore’s Law, more room on die for bigger L1$ and for second-level L2$
    – And in some cases even an L3$!
• Mainframes have ~1GB L4 cache off-chip
Break!
Outline

• Understanding Cache Misses
• Increasing Cache Performance
• Performance of Multi-level Caches (L1, L2, ...)
• Real world example caches
• And in Conclusion ...
Memory Hierarchy

As we move to outer levels the latency goes up and price per bit goes down.
Local vs. Global Miss Rates

- **Local miss rate**: fraction of references to a given level of a cache that miss
  - Local Miss rate $L2$ = $L2$ Misses / $L1$ Misses
    = $L2$ Misses / total_L2_accesses

- **Global miss rate**: the fraction of references that miss in all levels of a multilevel cache and go all the way to memory
  - $L2$ local miss rate $>>$ than the global miss rate
Local vs. Global Miss Rates

- **Local miss rate** – the fraction of references to one level of a cache that miss
  - Local Miss rate L2$ = $L2 Misses / L1$ Misses

- **Global miss rate** – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate

- Global Miss rate = L2$ Misses / Total Accesses
  = (L2$ Misses / L1$ Misses) × (L1$ Misses / Total Accesses)
  = Local Miss rate L2$ × Local Miss rate L1$

- AMAT = Time for a hit + Miss rate × Miss penalty
- AMAT = Time for a L1$ hit + (local) Miss rate L1$ ×
  (Time for a L2$ hit + (local) Miss rate L2$ × L2$ Miss penalty)
Multilevel Cache Considerations

• Different design considerations for L1$ and L2$
  – L1$ focuses on **fast access**: minimize hit time to achieve shorter clock cycle, e.g., smaller $
  – L2$, L3$ focus on **low miss rate**: reduce penalty of long main memory access times: e.g., 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 even with higher miss rate

• For the L2$, fast hit time is less important than low miss rate
  – L2$ hit time determines L1$’s miss penalty
  – L2$ local miss rate >> than the global miss rate
FIGURE 5.47 The L1, L2, and L3 data cache miss rates for the Intel Core i7 920 running the full integer SPECCPU2006 benchmarks.
Outline

• Understanding Cache Misses
• Increasing Cache Performance
• Performance of Multi-level Caches (L1, L2, ...)
• Real world example caches
• And in Conclusion ...
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Intel Nehalem</th>
<th>AMD Opteron X4 (Barcelona)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>Split instruction and data caches</td>
<td>Split instruction and data caches</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KB each for instructions/data per core</td>
<td>64 KB each for instructions/data per core</td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L1 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L1 hit time (load-use)</td>
<td>Not Available</td>
<td>3 clock cycles</td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>Unified (instruction and data) per core</td>
<td>Unified (instruction and data) per core</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>256 KB (0.25 MB)</td>
<td>512 KB (0.5 MB)</td>
</tr>
<tr>
<td>L2 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L2 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L2 hit time</td>
<td>Not Available</td>
<td>9 clock cycles</td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>Unified (instruction and data)</td>
<td>Unified (instruction and data)</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>8192 KB (8 MB), shared</td>
<td>2048 KB (2 MB), shared</td>
</tr>
<tr>
<td>L3 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L3 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L3 hit time</td>
<td>Not Available</td>
<td>38 (?)clock cycles</td>
</tr>
</tbody>
</table>
## CPI/Miss Rates/DRAM Access

### SpecInt2006

<table>
<thead>
<tr>
<th>Name</th>
<th>CPI</th>
<th>L1 D cache misses/1000 instr</th>
<th>L2 D cache misses/1000 instr</th>
<th>DRAM accesses/1000 instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>0.75</td>
<td>3.5</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>bzip2</td>
<td>0.85</td>
<td>11.0</td>
<td>5.8</td>
<td>2.5</td>
</tr>
<tr>
<td>gcc</td>
<td>1.72</td>
<td>24.3</td>
<td>13.4</td>
<td>14.8</td>
</tr>
<tr>
<td>mcf</td>
<td>10.00</td>
<td>106.8</td>
<td>88.0</td>
<td>88.5</td>
</tr>
<tr>
<td>go</td>
<td>1.09</td>
<td>4.5</td>
<td>1.4</td>
<td>1.7</td>
</tr>
<tr>
<td>hmer</td>
<td>0.80</td>
<td>4.4</td>
<td>2.5</td>
<td>0.6</td>
</tr>
<tr>
<td>sjeng</td>
<td>0.96</td>
<td>1.9</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>libquantum</td>
<td>1.61</td>
<td>33.0</td>
<td>33.1</td>
<td>47.7</td>
</tr>
<tr>
<td>h264avc</td>
<td>0.80</td>
<td>8.8</td>
<td>1.6</td>
<td>0.2</td>
</tr>
<tr>
<td>omnetpp</td>
<td>2.94</td>
<td>30.9</td>
<td>27.7</td>
<td>29.8</td>
</tr>
<tr>
<td>astart</td>
<td>1.79</td>
<td>16.3</td>
<td>9.2</td>
<td>8.2</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>2.70</td>
<td>38.0</td>
<td>15.8</td>
<td>11.4</td>
</tr>
<tr>
<td>Median</td>
<td>1.35</td>
<td>13.6</td>
<td>7.5</td>
<td>5.4</td>
</tr>
</tbody>
</table>
Skylark: Intel’s Recent Generation

### Desktop processors

**"Skylake-X" (14 nm)**

- All models support: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AVX2, AVX-512, FMA3, MPX, Enhanced Intel SpeedStep Technology (EIST), Intel 64, XD bit (an NX bit implementation), Intel VT-x, Intel VT-d, Turbo Boost, Hyper-threading, AES-NI, Intel TSX-NI, Smart Cache.
- PCI Express lanes: 44

<table>
<thead>
<tr>
<th>Model number</th>
<th>sSpec number</th>
<th>Cores (Threads)</th>
<th>Frequency</th>
<th>Turbo Boost All-Core/2.0 (Max 3.0)</th>
<th>L2 cache</th>
<th>L3 cache</th>
<th>TDP</th>
<th>Socket</th>
<th>I/O bus</th>
<th>Memory</th>
<th>Release date</th>
<th>Part number(s)</th>
<th>Release price (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core i9-7900X</td>
<td>SR3L2 (U0)</td>
<td>10 (20)</td>
<td>3.3 GHz</td>
<td>4.0 GHz/4.3 GHz</td>
<td>10 x 1024 KiB</td>
<td>13.75 MiB</td>
<td>140 W LGA 2066 DMI 3.0</td>
<td>4 x DDR4-2666</td>
<td>June 2017</td>
<td>BX80673I97900X</td>
<td>CD806730328804</td>
<td>$989</td>
<td></td>
</tr>
<tr>
<td>Core i9-7920X</td>
<td>SR3NG (U0)</td>
<td>12 (24)</td>
<td>2.9 GHz</td>
<td>3.8 GHz/4.3 GHz</td>
<td>12 x 1024 KiB</td>
<td>16.50 MiB</td>
<td>140 W LGA 2066 DMI 3.0</td>
<td>4 x DDR4-2666</td>
<td>August 2017</td>
<td>BX80673I97920X</td>
<td>CD806730375300</td>
<td>$1189</td>
<td></td>
</tr>
<tr>
<td>Core i9-7940X</td>
<td>SR3RQ (U0)</td>
<td>14 (28)</td>
<td>3.1 GHz</td>
<td>3.8 GHz/4.3 GHz</td>
<td>14 x 1024 KiB</td>
<td>19.25 MiB</td>
<td>165 W LGA 2066 DMI 3.0</td>
<td>4 x DDR4-2666</td>
<td>September 2017</td>
<td>BX80673I97940X</td>
<td>CD806730374701</td>
<td>$1399</td>
<td></td>
</tr>
<tr>
<td>Core i9-7960X</td>
<td>SR3RR (U0)</td>
<td>16 (32)</td>
<td>2.8 GHz</td>
<td>3.6 GHz/4.2 GHz</td>
<td>16 x 1024 KiB</td>
<td>22.00 MiB</td>
<td>165 W LGA 2066 DMI 3.0</td>
<td>4 x DDR4-2666</td>
<td>September 2017</td>
<td>BX80673I97960X</td>
<td>CD8067303734802</td>
<td>$1699</td>
<td></td>
</tr>
<tr>
<td>Core i9-7980X</td>
<td>SR3RS (U0)</td>
<td>18 (36)</td>
<td>2.6 GHz</td>
<td>3.4 GHz/4.2 GHz</td>
<td>18 x 1024 KiB</td>
<td>24.75 MiB</td>
<td>165 W LGA 2066 DMI 3.0</td>
<td>4 x DDR4-2666</td>
<td>September 2017</td>
<td>BX80673I97800X</td>
<td>CD8067303734902</td>
<td>$1999</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Understanding Cache Misses
• Increasing Cache Performance
• Performance of Multi-level Caches (L1, L2, ...)
• Real world example caches
• And in Conclusion ...
Bottom Line: Cache Design Space

• 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)
  – Depends on technology / cost

• Simplicity often wins