CS 61C:
Great Ideas in Computer Architecture
More Cache: Set Associativity

Instructor:
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/sp12

Review

• Big Ideas of Instruction-Level Parallelism
• Pipelining, Hazards, and Stalls
• Forwarding, Speculation to overcome Hazards
• Multiple issue to increase performance
  – IPC instead of CPI
• Dynamic Execution: Superscalar in-order issue, branch prediction, register renaming, out-of-order execution, in-order commit
  – “unroll loops in HW”, hide cache misses

Agenda

• Cache Memory Recap
• Administrivia
• Set-Associative Caches
• AMAT and Multilevel Cache Review
• Nehalem Memory Hierarchy

Recap: Components of a Computer

Recap: Typical Memory Hierarchy

• Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology
Recap: Cache Performance and Average Memory Access Time (AMAT)

- CPU time = IC \times CPI \times CC
  = IC \times (CPI_{ideal} + \text{Memory-stall cycles}) \times CC

Memory-stall cycles = \text{Read-stall cycles} + \text{Write-stall cycles}

Read-stall cycles = \text{reads/program} \times \text{read miss rate} \times \text{read miss penalty}

Write-stall cycles = \text{writes/program} \times \text{write miss rate} \times \text{write miss penalty}

- AMAT is the average time to access memory considering both hits and misses
  AMAT = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}

Improving Cache Performance

- Reduce the time to hit in the cache
  - E.g., Smaller cache, direct-mapped cache, special tricks for handling writes

- Reduce the miss rate
  - E.g., Bigger cache, larger blocks
  - More flexible placement (increase associativity)

- Reduce the miss penalty
  - E.g., Smaller blocks or critical word first in large blocks, special tricks for handling writes, faster/higher bandwidth memories
  - Use multiple cache levels

Sources of Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - 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 (may increase access time)

Reducing Cache Misses

- Allow more flexible block placement

- Direct mapped $\rightarrow$: memory block maps to exactly one cache block

- Fully associative $\rightarrow$: allow a memory block to be mapped to any cache block

- Compromise: divide $\rightarrow$ into sets, each of which consists of n "ways" (n-way set associative) to place memory block
  - Memory block maps to unique set determined by index field and is placed in any of the n-ways of that set
  - Calculation: (block address) modulo (n sets in the cache)

Alternative Block Placement Schemes

- Direct mapped
- Set associative
- Fully associative

Projects

- Project 4: Pipelined Cycle Processor in Logisim
  - Due 4/15
- Extra Credit: Fastest Version of Project 3
  - Due 4/22 11:59 PM
- All grades finalized: 4/27
- Final Review: Sunday April 29, 2-5PM, 2050 VLSB
- Extra office hours: Thu-Fri May 3 and May 4
- Final: Wed May 9 11:30-2:30, 1 PIMENTEL

Administrivia
Get to Know Your Prof

- Learn your genealogy (before it's too late to ask)
- Pattersons go to Penn.
- Our church was on the Underground Railroad — My great-grandfather named after the church minister
- John Patterson joins Union Army after Emancipation Proclamation

Agenda

- Cache Memory Recap
- Administrivia
- Set-Associative Caches
- AMAT and Multilevel Cache Review
- Nehalem Memory Hierarchy

Example: 4-Word Direct-Mapped $Worst-Case Reference String$

- Consider the main memory word reference string
- Start with an empty cache - all blocks initially marked as not valid

Example: 4-Word Direct-Mapped $Worst-Case Reference String$

- Consider the main memory word reference string
- Start with an empty cache - all blocks initially marked as not valid
- 8 requests, 8 misses
- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

Example: 2-Way Set Associative $ (4 words = 2 sets x 2 ways per set)$

- Main Memory
- One word blocks
- Two low order bits define the byte in the word (32b words)
- Q: How do we find it?
- Use next 3 low order memory address bits to determine which cache set (i.e., modulo the number of sets in the cache)

Example: 4 Word 2-Way SA $Same Reference String$

- Consider the main memory word reference string
- Start with an empty cache - all blocks initially marked as not valid
Example: 4-Word 2-Way SA $\$
Same Reference String
• Consider the main memory word reference string
  Start with an empty cache - all blocks
  initially marked as not valid
  4/12/11

  • 8 requests, 2 misses
  • Solves the ping pong effect in a direct mapped cache due to
    conflict misses since now two memory locations that map into
    the same cache set can co-exist!

  000 Mem(0)
  000 Mem(0)
  000 Mem(0)
  000 Mem(0)
  000 Mem(0)
  000 Mem(0)
  000 Mem(0)
  000 Mem(0)

Example: Eight-Block Cache with
Different Organizations

Four-Way Set-Associative Cache
• $2^3 = 256$ sets each with four ways
  (each with one block)

Range of Set-Associative Caches
• For a fixed-size cache, 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

Range of Set-Associative Caches
• For a fixed-size cache, 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

Costs of Set-Associative Caches
• When miss occurs, which way’s block selected for
  replacement?
  – Least Recently Used (LRU): one that has been unused the
    longest
    • Must track when each way’s block was used relative to other
      blocks in the set
    • For 2-way SA $S$, one bit per set $\rightarrow$ set to 1 when a block is
      referenced; reset the other way’s bit (i.e., “last used”)
  – 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 $S$ 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
Cache Block Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache item and throw it out
- Least Recently Used
  - Hardware keeps track of access history
  - 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

Benefits of Set-Associative Caches

- Choice of DM 5 or SA 5 depends on the cost of a miss versus the cost of implementation
- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

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**: Chance 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

Reduce AMAT

- Use multiple levels of cache
- As technology advances, more room on IC die for larger L1$ or for additional levels of cache (e.g., L2$ and L3$)
- Normally the higher cache levels are unified, holding both instructions and data

AMAT Revisited

- For a 2nd-level cache, L2 Equations:
  \[
  \text{AMAT} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}
  \]
- Definitions:
  - *Local miss rate*: misses in this $S$ divided by the total number of memory accesses to this $S$ (Miss rate$_{L2}$)
  - *Global miss rate*: misses in this $S$ divided by the total number of memory accesses generated by the CPU (Miss Rate$_{L2}$ x Miss Rate$_{L1}$)
  - *Global miss rate* is what matters to overall performance
  - Local miss rate is factor in evaluating the effectiveness of L2 cache
CPI\textsubscript{stalls} Calculation

• Assume
  – CPI\textsubscript{ideal} of 2
  – 100 cycle miss penalty to main memory
  – 25 cycle miss penalty to Unified L2S
  – 36% of instructions are load/stores
  – 2% L1 $I$ miss rate; 4% L1 D$S$ miss rate
  – 0.5% $U$ (unified) L2S miss rate

\[
\text{CPI}_{\text{stalls}} = 2 + \frac{1 \times 0.02 \times 25}{L1} + \frac{0.36 \times 0.04 \times 25}{L2} = 3.54 \text{ (vs. 5.44 with no L2S)}
\]

AMAT Calculations

Local vs. Global Miss Rates

Example:

• For 1000 memory refs:
  – 40 misses in L1$S$ (miss rate 4%)
  – 20 misses in L2$S$ (miss rate 2%)
  – L1$S$ hits 1 cycle,
  – L2$S$ hits in 10 cycles
  – Miss to MM costs 100 cycles

• 1.5 memory references per instruction (i.e., 50% Ld/St)
  – 1000 mem refs = 667 intrs,
  – OR 1000 intrs = 1500 mem refs

Ask:

Local miss rate

AMAT

Stall cycles per instruction with and without L2S

With L2S

• Local miss rate =
  – AMAT
  – Ave Mem Stalls/Ref
  – Ave Mem Stalls/Instr

Without L2S

• AMAT
  – Ave Mem Stalls/Ref
  – Ave Mem Stalls/Instr

Assume ideal CPI=1.0,

performance improvement =

CPI/Miss Rates/DRAM Access

SpecInt2006

<table>
<thead>
<tr>
<th>Name</th>
<th>CPI</th>
<th>L1 $D$ cache memory access</th>
<th>L2 $D$ cache memory access</th>
<th>DRAM access latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>0.79</td>
<td>2.9</td>
<td>1.1</td>
<td>1.9</td>
</tr>
<tr>
<td>blaze</td>
<td>0.80</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>ref</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>himek</td>
<td>0.80</td>
<td>4.4</td>
<td>2.5</td>
<td>0.8</td>
</tr>
<tr>
<td>spam</td>
<td>0.96</td>
<td>1.9</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>rahtnc</td>
<td>1.61</td>
<td>33.0</td>
<td>33.1</td>
<td>47.7</td>
</tr>
<tr>
<td>nhstore</td>
<td>0.80</td>
<td>8.8</td>
<td>3.6</td>
<td>0.2</td>
</tr>
<tr>
<td>onenlp</td>
<td>2.94</td>
<td>30.9</td>
<td>27.7</td>
<td>29.8</td>
</tr>
<tr>
<td>atari</td>
<td>1.79</td>
<td>16.3</td>
<td>9.2</td>
<td>8.2</td>
</tr>
<tr>
<td>valanc</td>
<td>2.10</td>
<td>38.0</td>
<td>15.5</td>
<td>11.4</td>
</tr>
<tr>
<td>mtkns</td>
<td>3.25</td>
<td>13.6</td>
<td>7.3</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Design Considerations

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

• Miss penalty of L1$S$ is significantly reduced by presence of L2$S$, so can be smaller/faster even with higher miss rate

• For the L2$S$, fast hit time is less important than low miss rate
  – L2$S$ hit time determines L1$S$’s miss penalty
  – L2$S$ local miss rate >> than the global miss rate

Improving Cache Performance

• Reduce the time to hit in the cache
  – E.g., Smaller cache, direct-mapped cache, special tricks for handling writes

• Reduce the miss rate (in L2$S$, L3$S$)
  – E.g., Bigger cache, larger blocks
  – More flexible placement (increase associativity)

• Reduce the miss penalty (in L1$S$)
  – E.g., Smaller blocks or critical word first in large blocks, special tricks for handling for writes, faster/ higher bandwidth memories
  – Use multiple cache levels
Sources of Cache Misses: 3Cs for L2S, L3S

- **Compulsory** (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - 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 (may increase access time)

Two Machines’ Cache Parameters

<table>
<thead>
<tr>
<th></th>
<th>Intel Nehalem</th>
<th>AMD Barcelona</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 cache</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>organization &amp; size</td>
<td>Split I$ and D$; 32KB for each core, 64B blocks</td>
<td>Split I$ and D$, 64KB for each core, 64B blocks</td>
</tr>
<tr>
<td><strong>L1 write policy</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
<tr>
<td><strong>L2 cache</strong></td>
<td>Unified; 256KB (0.25MB) per core, 64B blocks</td>
<td>Unified; 512KB (0.5MB) per core, 64B blocks</td>
</tr>
<tr>
<td><strong>L2 write policy</strong></td>
<td>write-back</td>
<td>write-back</td>
</tr>
<tr>
<td><strong>L2 cache</strong></td>
<td>unified, 8192KB (8MB) shared by cores, 64B blocks</td>
<td>unified, 2048KB (2MB) shared by cores, 64B blocks</td>
</tr>
<tr>
<td><strong>L2 associativity</strong></td>
<td>8-way set assoc., ~LRU</td>
<td>16-way set assoc., ~LRU</td>
</tr>
<tr>
<td><strong>L3 cache</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>L3 write policy</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
<tr>
<td><strong>L3 cache</strong></td>
<td>unified, 8192KB (8MB) shared by cores, 64B blocks</td>
<td>unified, 2048KB (2MB) shared by cores, 64B blocks</td>
</tr>
<tr>
<td><strong>L3 associativity</strong></td>
<td>16-way set assoc.</td>
<td>32-way set assoc., evict block shared by fewest cores</td>
</tr>
<tr>
<td><strong>L3 write policy</strong></td>
<td>write-back, write-allocate</td>
<td>write-back, write-allocate</td>
</tr>
</tbody>
</table>

Nehalem Memory Hierarchy Overview

Cache Hierarchy Latencies

- **L1 I & D** 32KB 8-way, latency 4 cycles, 64B blocks
- **L2** 256 KB 8-way, latency <12 cycles
- **L3** 8 MB, 16-way, latency 30-40 cycles
- **DRAM**, latency ~180-200 cycles

Core’s Private Memory System
All Sockets can Access all Data

-~60ns

Local Memory Access

How ensure that get data allocated to local DRAM?

Such systems called "NUMA" for Non-uniform Memory Access: some addresses are slower than others

-~100ns

Remote Memory Access

Why Inclusive?

- Inclusive cache provides benefit of on-die snoop filter
- Core Valid Bits
  - 1 bit per core per cache line
  - If line may be in a core, set core valid bit
  - Snoop only needed if line is in L3 and core valid bit is set
- Scalability
  - Addition of cores/sockets does not increase snoop traffic seen by cores
- Latency
  - Minimize effective cache latency by eliminating cross-core snoops in the common case
  - Minimize snoop response time for cross-socket cases

Non-Uniform Memory Access (NUMA)

- FSB architecture
  - All memory in one location
  - Starting with Intel® Core™ microarchitecture (Nehalem)
    - Memory located in multiple places
  - Latency to memory dependent on location
  - Local memory has highest BW, lowest latency
  - Remote Memory still very fast

Ensure software is NUMA-optimized for best performance

Intel® Smart Cache – 3rd Level Cache

- Shared across all cores
- Size depends on # of cores
  - Quad-core: Up to 8MB (16-ways)
  - Scalability:
    - Built to vary size with varied core counts
    - Built to easily increase L3 size in future parts
- Perceived latency depends on frequency ratio between core & uncore
- Inclusive cache policy for best performance
  - Address residing in L1/L2 must be present in 3rd level cache

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 (L1-cache, D-cache)
    - Depends on technology / cost
  - Simplicity often wins

Intel® Smart Cache – 3rd Level Cache
Summary

- Name of the Game: Reduce Cache Misses
  - 2 memory blocks mapping to same block knock each other out as program bounces from 1 memory location to next
- One way to do it: set-associativity
  - Memory block maps into more than 1 cache block
  - N-way: n possible places in cache to hold a memory block
- N-way Cache of $2^{N+M}$ blocks: $2^N$ ways x $2^M$ sets
- Multi-level caches
  - Optimize first level to be fast!
  - Optimize 2nd and 3rd levels to minimize the memory access penalty