Lectures 16
Cache

Pramod V. Argade
May 23, 2005
Announcements

- **Reading Assignment**
  Chapter 7: The Basics of Caches
  - 7.1, 7.2

- **Homework 6: Due Wednesday May 25th**
  - 6.31, 6.32
  - 7.2, 7.3, 7.4, 7.5, 7.6, 7.9, 7.10, 7.12, 7.23

- **Quiz**
  **When:** Wed., May 25th, First 10 minutes of the class
  **Topic:** Cache Chapter 7  **Need:** Paper, pen

Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/28</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3/30</td>
<td>Wednesday</td>
<td>Performance, Ch. 4</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>4/4</td>
<td>Monday</td>
<td>ISA, Ch. 2</td>
<td>Performance #1</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4/6</td>
<td>Wednesday</td>
<td>Arithmetic, Ch. 3</td>
<td>ISA #2</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>4/11</td>
<td>Monday</td>
<td>Arithmetic, Ch. 3</td>
<td>ISA #2</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>4/13</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>4/18</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Arithmetic #3</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>4/20</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>4/25</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU #4</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>4/27</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>5/2</td>
<td>Monday</td>
<td>Mid-term Exam Center 101</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>5/5</td>
<td>Monday</td>
<td>Exceptions, Ch. 5 and Pipeline, Ch. 6</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>5/11</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>5/16</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards #5</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>5/18</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>5/23</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>5/25</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>Cache #6</td>
<td>-</td>
</tr>
<tr>
<td>No Class</td>
<td>5/30</td>
<td>Monday</td>
<td>Memorial Day Holiday</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>6/1</td>
<td>Wednesday</td>
<td>Course Review</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>6/10</td>
<td>Friday</td>
<td>7 - 10 PM, SOLIS 107: Final Exam</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Dealing with Stores

- Stores must be handled differently than loads, because...
  - They don’t necessarily require the CPU to stall
    - stores don’t produce register values used by other instructions
  - They change the content of cache/memory (creating memory consistency issues)

- Policy decisions for stores
  - write-through => all writes go to both cache and main memory
  - write-back => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
  - write-allocate => on a store miss, bring written line into the cache
  - write-around => on a store miss, write to main memory, & ignore cache

Handling a Cache Write Miss

- Write-through Cache
  - Write data to cache as well as memory
  - Don’t need to consider whether the write hits or misses the cache
  - Disadvantage: Every write causes the data to be written to the main memory
    - Use write buffer so CPU can proceed with the following instructions

- Write-back Cache
  - When write occurs, write the new value only to the block in the cache
  - Write cache data to memory when it is about to be overwritten for another address
  - Improves performance over write-through cache
  - More complex to implement
Summary for Stores

- On a store hit, write the new data to cache. In a write-through cache, write the data immediately to memory. In a write-back cache, mark the line as dirty.
- On a store miss, initiate a cache block load from memory for a write-allocate cache. Write directly to memory for a write-around cache.
- On any kind of cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.

Taking advantage of Spatial Locality

- Consider following address trace:
  - 0, 1, 2, 3, 17, 8, 9, 10, 11, 17, 4, 5, 6, 7…
- Notice that addresses lie in the vicinity of each other
- Instructions show high degree of spatial locality
  - Typically accessed sequentially
  - Generally, code consists of a lot of loops
- Data also shows spatial locality
  - Typically less than that of instructions
  - Different elements of a structure may be accessed
- Why not bring multiple words on a cache miss?
  - Instead of bringing a single?
Spatial Locality: Larger Cache Blocks

- Large cache blocks take advantage of spatial locality.
- Too large of a block size can waste cache space.
- Larger cache blocks require less tag space.

A 64 KB Cache using 16-byte Blocks

- 4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).
Complication with Larger Blocks

- Write-through cache: Can’t write to the cache while performing a tag comparison
  - Ok if there is a hit in the cache
  - Not ok if there is a cache miss:
    - The block has to be fetched from memory and placed in the cache
    - Rewrite the word that caused the miss into the cache

Impact of Block Size on Miss Rate

- In general, larger block decreases miss rate, however,
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
    - Miss rates go up if block size is too big
    - Since there are too few cache blocks
Cache Performance

- 64 KB each instruction cache and data cache (direct mapped)

In general, Average Access Time:

\[ \text{Average Access Time} = \text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate} \]

Limitations of direct mapped cache
- A block can go in exactly one place in the cache

<table>
<thead>
<tr>
<th>Program</th>
<th>Block Size in words</th>
<th>Instruction miss rate</th>
<th>Data miss rate</th>
<th>Effective Combined miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>1</td>
<td>6.1%</td>
<td>2.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.9%</td>
</tr>
<tr>
<td>spice</td>
<td>1</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.3%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

Flexible Placement of Blocks

- Direct mapped cache
  - A block can go in exactly one place in the cache
  - Leads to collision among blocks

- Fully associative cache
  - A block can go in any place in the cache
  - All addresses have to be compared simultaneously
  - Slow and expensive

- N-way set-associative cache
  - Consists of a number of sets
  - Each set consists of N blocks
  - Each block in memory maps to a unique set
    - A block can be placed in any element of the set
Locating a Block in a Cache

Block address = $12_{10}$

Direct Mapped

Set-associative

Fully associative

Block # 0 1 2 3 4 5 6 7

Search

Set # 0 1 2 3

Search

Tag

Data

Tag

Data

Tag

Data

Cache Configurations

- An eight-block cache with various configurations

Eight-way set associative (fully associative)

Two-way set associative

Four-way set associative

Right-way set associative (fully associative)

Which cache has the largest number of tag bits?

Which cache has the least?
Accessing a 4-way Set-associative cache

- 4 K-byte 4-way set-associative cache, with a block size of 4 bytes

Accessing a Direct Mapped Cache

- 64 KB cache, direct-mapped, 32-byte cache block size
Accessing a Set-associative Cache

- 32 KB cache, 2-way set-associative, 16-byte block size

Cache Associativity and Miss Rate

Data cache miss rate for SPEC2000 benchmark

- Benefit of going from direct mapped to two-way set associative cache is significant.
- Benefits of further increase in associativity are smaller.
Associative Caches: Higher hit rates, but...

- Longer access time (longer to determine hit/miss, more muxing of outputs)
- More space (longer tags)
  - 16 KB, 16-byte blocks, direct mapped, tag = ?
  - 16 KB, 16-byte blocks, 4-way, tag = ?

A Fully-associative cache

Word address string:

| 20 | 00010100 |
| 5  | 00000101 |
| 10 | 00001010 |
| 12 | 00001100 |
| 4  | 00000100 |
| 9  | 00001001 |
| 7  | 00000111 |
| 8  | 00001000 |
| 21 | 00010101 |
| 24 | 00011000 |
| 14 | 00001110 |
| 11 | 00001011 |
| 4  | 00000100 |

The tag identifies the address of the cached data
Valid bit indicates that entry is valid

- A cache that can put a block of data anywhere is called _fully associative_
- To access the cache, address must be compared with all the entries in the cache

4 entries, each block holds one word, any block can hold any word.
Cache Organization

- A typical cache has three dimensions

The Three Cs

- **Compulsory misses:** First time access
  - Caused by the first access to a block that has never been in the cache
  - Also called cold-start misses
  - Can be reduced by increasing the block size

- **Capacity misses:** Cache is not large enough
  - Caused when cache cannot contain all the blocks needed
  - Occur because of blocks being replaced and later retrieved
  - Can be reduced by enlarging the cache

- **Conflict misses:** Multiple addresses map to the same cache line
  - Occur in direct mapped and set-associative caches
  - Multiple blocks compete for the same set
  - Can be eliminated by using fully associative cache
Which Block Should be Replaced on a Miss?

- Direct Mapped is Easy
- Set associative or fully associative:
  - Random (large associativities)
  - LRU (smaller associativities)
- Miss rates for the two schemes:

<table>
<thead>
<tr>
<th></th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associativity:</td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.18%</td>
<td>5.69%</td>
<td>4.67%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.88%</td>
<td>2.01%</td>
<td>1.54%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>

LRU is preferred scheme for a small size cache

Measuring Cache Performance

\[
\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory stall cycles})
\]

* Clock cycle time

Memory stall cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = (Reads per program) * Read miss rate * Read miss penalty

Write-stall cycles = (Writes per program) * Write miss rate * Write miss penalty

Pramod Argade
UCSD CSE 141, Spring 2005
Cache Performance

- Ideal CPI for a processor (i.e. without memory stalls) is 1.4.
- 35% of the instructions are loads and stores.
- Miss penalty is 10 cycles.
- How much do we improve the performance if the miss rate is reduced from 10% to 2%?

Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
  - Three Major Categories of Cache Misses:
    - Compulsory Misses: sad facts of life. Example: cold start misses.
    - Conflict Misses: increase cache size and/or associativity.
      Nightmare Scenario: ping pong effect!
    - Capacity Misses: increase cache size
  - Cache Design Space
    - total size, block size, associativity
    - replacement policy
    - write-hit policy (write-through, write-back)
  - Caches give an illusion of a large, cheap memory with the access time of a fast, expensive memory