Caches (2)

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Outline

Textbook: P&H 5.2 (cont) – 5.3 (4th Ed.) / P&H 5.3 (cont) – 5.4 (5th Ed.)

• Write Policies
• Cache Performance
• Associative Caches
• Multilevel Caches
• Cache’s Interaction with Advanced CPUs and Software
Cache Misses

• On cache hit, CPU proceeds normally

• On cache miss
  – Stall the CPU pipeline
  – Fetch block from next level of hierarchy
  – Instruction cache miss
    • Restart instruction fetch
  – Data cache miss
    • Complete data access
Write Policies (1)

• Write-Through
  – On data-write hit, could just update the block in cache
  • But then cache and memory would be inconsistent
  – Write through: also update memory
  – But makes writes take longer
    • e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
      – Effective CPI = 1 + 0.1×100 = 11
  – Solution: write buffer
    • Holds data waiting to be written to memory
    • CPU continues immediately
      – Only stalls on write if write buffer is already full
Write Policies (2)

• Write-Back
  – Alternative: On data-write hit, just update the block in cache
    • Keep track of whether each block is dirty
  – When a dirty block is replaced
    • Write it back to memory
    • Can use a write buffer to allow replacing block to be read first
Write Policies (3)

• Write Allocation
  – What should happen on a write miss?
  – Alternatives for write-through
    • Allocate on miss: fetch the block
    • Write around: don’t fetch the block
      – Since programs often write a whole block before reading it (e.g., initialization)
  – For write-back
    • Usually fetch the block
Example: Intrinsity FastMATH

• Embedded MIPS processor
  – 12-stage pipeline
  – Instruction and data access on each cycle

• Split cache: separate I-cache and D-cache
  – Each 16KB: 256 blocks × 16 words/block
  – D-cache: write-through or write-back

• SPEC2000 miss rates
  – I-cache: 0.4%
  – D-cache: 11.4%
  – Weighted average: 3.2%
Example: Intrinsity FastMATH
Cache Performance (I)

- **Measuring Cache Performance**
  - Components of CPU time
    - Program execution cycles
      - Includes cache hit time
    - Memory stall cycles
      - Mainly from cache misses
  - With simplifying assumptions:
    
    \[
    \text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}
    \]
    
    \[
    = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}
    \]
Cache Performance (2)

• Example:
  – I-cache miss rate = 2%
  – D-cache miss rate = 4%
  – Miss penalty = 100 cycles
  – Base CPI (ideal cache) = 2
  – Load & stores are 36% of instructions

• Miss cycles per instruction
  – I-cache: $0.02 \times 100 = 2$
  – D-cache: $0.36 \times 0.04 \times 100 = 1.44$

• Actual CPI = $2 + 2 + 1.44 = 5.44$
  – Ideal CPU is $5.44/2 = 2.72$ times faster
Cache Performance (3)

• Average Access Time
  – Hit time is also important for performance
  – Average memory access time (AMAT)

\[
AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}
\]

– Example
  • CPU with 1ns clock
  • hit time = 1 cycle
  • miss penalty = 20 cycles
  • L-cache miss rate = 5%
  • AMAT = 1 + 0.05 \times 20 = 2ns
    – 2 cycles per instruction
Cache Performance (4)

• Performance summary
  – When CPU performance increased
    • Miss penalty becomes more significant
  – Decreasing base CPI
    • Greater proportion of time spent on memory stalls
  – Increasing clock rate
    • Memory stalls account for more CPU cycles
  – Can’t neglect cache behavior when evaluating system performance
Associative Caches (I)

• Fully associative
  – Allow a given block to go in any cache entry
  – Requires all entries to be searched at once
  – Comparator per entry (expensive)

• \( n \)-way set associative
  – Each set contains \( n \) entries
  – Block number determines which set
  • (Block number) modulo (#Sets in cache)
  – Search all entries in a given set at once
  – \( n \) comparators (less expensive)
Associative Cache (2)

• Example

**Direct mapped**

<table>
<thead>
<tr>
<th>Block #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Set associative**

<table>
<thead>
<tr>
<th>Set #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fully associative**

<table>
<thead>
<tr>
<th>Data</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Associative Cache (3)

- Spectrum of associativity: for a cache with 8 entries

One-way set associative
(direct mapped)

<table>
<thead>
<tr>
<th>Block</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Four-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Eight-way set associative (fully associative)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Associative Cache (4)

• Associativity example: compare 4-block caches
  – Direct mapped, 2-way set associative, fully associative
  – Block access sequence: 0, 8, 0, 6, 8

• Direct mapped

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
</tbody>
</table>
## Associative Cache (5)

- **2-way set associative**

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hit</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
</tbody>
</table>

- **Fully associative**

<table>
<thead>
<tr>
<th>Block address</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Set 0</td>
</tr>
<tr>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>0</td>
<td>hit</td>
<td>Mem[0]</td>
</tr>
</tbody>
</table>
Associative Cache (6)

• How Much Associativity
  – Increased associativity decreases miss rate
    • But with diminishing returns
  – Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
    • 1-way: 10.3%
    • 2-way: 8.6%
    • 4-way: 8.3%
    • 8-way: 8.1%
Associative Cache (7)

- Set associative cache organization

![Diagram of set associative cache organization]
Replacement Policy

- **Direct mapped:** no choice

- **Set associative**
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set

- **Least-recently used (LRU)**
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that

- **Random**
  - Gives approximately the same performance as LRU for high associativity
Multilevel Caches (I)

• Multilevel caches
  – Primary cache attached to CPU
    • Small, but fast
  – Level-2 cache services misses from primary cache
    • Larger, slower, but still faster than main memory
  – Main memory services L2 cache misses
  – Some high-end systems include L3 cache
My Hardware: An Example

### Hardware Overview:

- **Model:** iMac
- **Model Identifier:** iMac15,1
- **Processor Name:** Intel Core i7
- **Processor Speed:** 4 GHz
- **Core Count:** 1
- **Core Count:** 4
- **L2 Cache:** 256 KB
- **L3 Cache:** 8 MB
- **RAM:** 32 GB
- **Boot ROM Version:** IM151.0207.B16
- **SMC Version:** 2.23f11
- **System BIOS:** C02NR5N4FY14
- **Hardware UUID:** 655F41BB-3CB1-539B-B1A4-72271A597056

---

ATA
Bluetooth
FireWire
NVMeExpress
PCI
SAS
SATA/SATA Express
SPI
Thunderbolt
USB
iBridge
Graphics/Display
Disk Drive
Memory
Battery SCSI
Audio
Network Card
Battery
Hard Drive
Card Reader
Camera
USB Channel
Printer
Hardware RAID
Multilevel Caches (2)

• Multilevel cache example
  – CPU base CPI = 1
  – Clock rate = 4GHz
  – Miss rate/instruction = 2%
  – Main memory access time = 100ns

• With just primary cache
  – Miss penalty = 100ns/0.25ns = 400 cycles
  – Effective CPI = 1 + 0.02 × 400 = 9
Multilevel Caches (3)

• Now add L2 cache
  – Access time = 5ns
  – Global miss rate to main memory = 0.5%

• Primary miss with L2 hit
  – Penalty = 5ns/0.25ns = 20 cycles

• Primary miss with L2 miss
  – Extra penalty = 500 cycles

• CPI = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4

• Performance ratio = 9/3.4 = 2.6
Multilevel Caches (4)

• Considerations
  – Primary cache
    • Focus on minimal hit time
  – L2 cache
    • Focus on low miss rate to avoid main memory access
    • Hit time has less overall impact

• Results
  • L1 cache usually smaller than a single cache
  • L1 block size smaller than L2 block size
  • L2 cache is much larger than in a single-level cache
  • L2 cache uses higher associativity for reducing miss rates
Interactions with Advanced CPUs

• Out-of-order CPUs can execute instructions during cache miss
  – Pending store stays in load/store unit
  – Dependent instructions wait in reservation stations
  • Independent instructions continue

• Effect of miss depends on program data flow
  – Much harder to analyze
  – Use system simulation
Interactions with Software

• Misses depend on memory access patterns
  – Algorithm behavior
  – Compiler optimization for memory access