Memory Hierarchies

Instructor: Dmitri A. Gusev

Fall 2007

CS 502: Computers and Communications Technology

Lecture 10, October 8, 2007
Memories

- **SRAM:**
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)

- **DRAM:**
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)
Exploiting Memory Hierarchy

- Users want large and fast memories!
  SRAM access times are $0.5 - 5\text{ ns}$ at cost of $\$4000$ to $\$10,000$ per GB.
  DRAM access times are $50-70\text{ ns}$ at cost of $\$100$ to $\$200$ per GB.
  Disk access times are $5$ to $20$ million $\text{ ns}$ at cost of $\$0.50$ to $\$2$ per GB.

- Try and give it to them anyway
  - build a memory hierarchy

![Diagram showing levels in the memory hierarchy with increasing distance from the CPU in access time and size of the memory at each level.](image)
An Expanded View of the Memory System

- Processor
  - Control
  - Datapath
    - Memory
    - Memory
  - Memory

- Memory

**Comparison**
- **Speed:** Fastest
- **Size:** Smallest
- **Cost:** Highest
  - Slowest
  - Biggest
  - Lowest
Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon.

Why does code have locality?

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
Memory Hierarchy: How Does it Work?

- Temporal Locality (Locality in Time):
  => Keep most recently accessed data items closer to the processor
- Spatial Locality (Locality in Space):
  => Move blocks consists of contiguous words to the upper levels
Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of
    
    \[ \text{RAM access time} + \text{Time to determine hit/miss} \]

- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level +
    
    Time to deliver the block the processor

- **Hit Time << Miss Penalty**
How is the hierarchy managed?

- Registers <-> Memory
  - by compiler (programmer?)
- cache <-> memory
  - by the hardware
- memory <-> disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)
Cache

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

E.g., lots of items at the lower level share locations in the upper level
Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache
Direct Mapped Cache

- For MIPS:

What kind of locality are we taking advantage of?
Direct Mapped Cache

- Taking advantage of spatial locality:
Hits vs. Misses

• Read hits
  – this is what we want!

• Read misses
  – stall the CPU, fetch block from memory, deliver to cache, restart

• Write hits:
  – can replace data in cache and memory (write-through)
  – write the data only into the cache (write-back the cache later)

• Write misses:
  – read the entire block into the cache, then write the word
Hardware Issues

• Make reading multiple words easier by using banks of memory

  a. One-word-wide memory organization

  b. Wide memory organization

  c. Interleaved memory organization

• It can get a lot more complicated...
Performance

• Increasing the block size tends to decrease miss rate:

• Use split caches because there is more spatial locality in code:

<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>
Performance

• Simplified model:

\[
\text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}
\]

\[
\text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}
\]

• Two ways of improving performance:
  – decreasing the miss ratio
  – decreasing the miss penalty

*What happens if we increase block size?*
Improving cache performance

Direct mapped

Block # 0 1 2 3 4 5 6 7
Data

Tag 1 2
Search

Set # 0 1 2 3
Data

Tag 1 2
Search

12 mod 8 = 4

Set associative

Fully associative

Set # 0
Data

Tag 1 2
Search

12 mod 4 = 0

Set # 1 2
Data

Tag 1 2
Search

12 mod 1 = 0
Compared to direct mapped, give a series of references that:

- results in a lower miss ratio using a 2-way set associative cache
- results in a higher miss ratio using a 2-way set associative cache assuming we use the “least recently used” replacement strategy

Cache size (blocks) = Number of sets * Associativity

Tag size increases as the associativity increases
### Example of associativity

#### Direct mapped cache

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index (block)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0 mod 4) = 0</td>
</tr>
<tr>
<td>6</td>
<td>(6 mod 4) = 2</td>
</tr>
<tr>
<td>8</td>
<td>(8 mod 4) = 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref.#</th>
<th>Block address</th>
<th>Hit or Miss</th>
<th>Cache index 0</th>
<th>Cache index 1</th>
<th>Cache index 2</th>
<th>Cache index 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Miss</td>
<td>Memory[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Miss</td>
<td>Memory[8]</td>
<td>Memory[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Miss</td>
<td>Memory[0]</td>
<td>Memory[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Miss</td>
<td>Memory[0]</td>
<td>Memory[0]</td>
<td>Memory[6]</td>
<td></td>
</tr>
</tbody>
</table>

#### 2-way associative

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index (set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0 mod 2) = 0</td>
</tr>
<tr>
<td>6</td>
<td>(6 mod 2) = 0</td>
</tr>
<tr>
<td>8</td>
<td>(8 mod 2) = 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref.#</th>
<th>Block address</th>
<th>Hit or Miss</th>
<th>Cache set 0</th>
<th>Cache set 0</th>
<th>Cache set 1</th>
<th>Cache set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Miss</td>
<td>Memory[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Miss</td>
<td>Memory[0]</td>
<td></td>
<td>Memory[0]</td>
<td>Memory[8]</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Hit</td>
<td>Memory[0]</td>
<td></td>
<td>Memory[0]</td>
<td>Memory[8]</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Miss</td>
<td>Memory[0]</td>
<td></td>
<td>Memory[0]</td>
<td>Memory[6]</td>
</tr>
</tbody>
</table>

#### Fully associative

<table>
<thead>
<tr>
<th>Ref.#</th>
<th>Block address</th>
<th>Hit or Miss</th>
<th>Cache block 0</th>
<th>Cache block 1</th>
<th>Cache block 2</th>
<th>Cache block 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Miss</td>
<td>Memory[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Miss</td>
<td>Memory[0]</td>
<td>Memory[0]</td>
<td>Memory[8]</td>
<td>Memory[8]</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Hit</td>
<td>Memory[0]</td>
<td>Memory[0]</td>
<td>Memory[8]</td>
<td>Memory[8]</td>
</tr>
</tbody>
</table>
An implementation
Performance

The graph shows the miss rate for different associativities and cache sizes. The y-axis represents the miss rate, ranging from 0% to 15%. The x-axis represents different associativities: One-way, Two-way, Four-way, and Eight-way. The cache sizes are indicated as 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, and 128 KB.

- One-way: The miss rate decreases from 15% to 0% as the cache size increases.
- Two-way: The miss rate decreases from 12% to 0% as the cache size increases.
- Four-way: The miss rate decreases from 9% to 0% as the cache size increases.
- Eight-way: The miss rate decreases from 6% to 0% as the cache size increases.

The graph indicates that larger cache sizes reduce the miss rate, which is beneficial for performance.
Decreasing miss penalty with multilevel caches

• Add a second level cache:
  – often primary cache is on the same chip as the processor
  – use SRAMs to add another cache above primary memory (DRAM)
  – miss penalty goes down if data is in 2nd level cache

• Example:
  – CPI of 1.0 on a 5 Ghz machine with a 5% miss rate, 100ns DRAM access
  – Adding 2nd level cache with 5ns access time decreases miss rate to .5%

• Using multilevel caches:
  – try and optimize the hit time on the 1st level cache
  – try and optimize the miss rate on the 2nd level cache
Cache Complexities

- Not always easy to understand implications of caches:

Theoretical behavior of Radix sort vs. Quicksort

Observed behavior of Radix sort vs. Quicksort
Cache Complexities

- Here is why:

  - Memory system performance is often critical factor
    - multilevel caches, pipelined processors, make it harder to predict outcomes
    - Compiler optimizations to increase locality sometimes hurt ILP

  - Difficult to predict best algorithm: need experimental data