Cache Memory

CPS 104
Lecture 17

Administrivia

- Homework #5 Due Tuesday
- Midterm II, Tuesday April 1
- Project Due Date: April 14, Midnight

Reading
- Chapter 7

Outline of Today’s Lecture

- Review
- The Memory Hierarchy
- Direct-mapped Cache
- Two-Way Set Associative Cache
- Fully Associative cache
- Replacement Policies
- Write Strategies

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer

Issues for Memory Systems

- Capacity/Size
- Cost
  - What technology is cheap?
- Performance
  - What technology is fast?
- Ease of Use
  - How much do programmers have to worry about it?

Cache

- What is a cache?
- What is the motivation for a cache?
- Why do caches work?
- How do caches work?
The Motivation for Caches

- **Motivation:**
  - Large memories (DRAM) are slow
  - Small memories (SRAM) are fast

  - Make the average access time small by:
    - Servicing most accesses from a small, fast memory.

  - Reduce the bandwidth required of the large memory

---

Levels of the Memory Hierarchy

- **Capacity:**
  - CPU Registers: 1-16 Bytes, <10ns
  - Cache: K Bytes, 10-100ns, $0.0005/bit
  - Main Memory: M Bytes, 100ns-1us, $0.0001/bit
  - Disk: G Bytes, ms, 10^-3 - 10^-4

- **Staging (After Hit):**
  - DS: 512-4K bytes
  - Uninitialized Memory

- **Capacity**
  - Upper Level (Cache)
  - Lower Level (Memory)

- **Access Time**
  - Upper Level (Cache)
  - Lower Level (Memory)

- **Cost**
  - Upper Level (Cache)
  - Lower Level (Memory)

- **Block:**
  - The minimum unit of information that can either be present or not present in the two level hierarchy

---

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 RAM access time + Time to determine hit/miss

- **Miss:** data needs to be retrieved from a 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 to the processor**

- **Hit Time << Miss Penalty**

---

Direct Mapped Cache

- **Direct Mapped cache is an array of fixed size blocks. Each block holds consecutive bytes of main memory data.**

- **The Tag Array holds the Block Memory Address.**

- **A valid bit associated with each cache block tells if the data is valid.**

  - **Cache Index:** The location of a block (and its tag) in the cache.
  - **Block Offset:** The byte location in the cache block.

  **Cache-Index = (<Address> Mod (Cache_Size)) / Block_Size**
  **Block-Offset = <Address> Mod (Block_Size)**
  **Tag = <Address> / (Cache_Size)**
The Simplest Cache: Direct Mapped Cache

- Memory Address
  - Location 0 can be occupied by data from:
    - Memory location 0, 4, 8, ... etc.
    - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> \( \Rightarrow \) cache index

Which one should we place in the cache?

How can we tell which one is in the cache?

Direct Mapped Cache (Cont.)

For a Cache of \( 2^M \) bytes with block size of \( 2^L \) bytes
- There are \( 2^{M-L} \) cache blocks,
- Lowest \( L \) bits of the address are Block-Offset bits
- Next \( (M - L) \) bits are the Cache-Index.
- The last \( (32 - M) \) bits are the Tag bits.

Example: 1-KB Cache with 32B blocks:
- Cache Index = \( \langle \text{Address} \rangle \mod (1024) \)/ 32
- Block-Offset = \( \langle \text{Address} \rangle \mod (32) \)
- Tag = \( \langle \text{Address} \rangle / (1024) \)

Example: 1KB Direct Mapped Cache with 32B Blocks
- For a 1024 \( (2^{10}) \) byte cache with 32-byte blocks:
  - The uppermost \( 22 = (32 - 10) \) address bits are the Cache Tag
  - The lowest 5 address bits are the Byte Select (Block Size = 2^5)
  - The next 5 address bits (bit5 - bit9) are the Cache Index

Example: 1K Direct Mapped Cache
- For a 1024 byte cache with 32-byte blocks:
  - The uppermost 22 address bits are the Cache Tag
  - The lowest 5 address bits are the Byte Select (Block Size = 2^5)
  - The next 5 address bits (bit5 - bit9) are the Cache Index

Example: 1K Direct Mapped Cache
- For a 1024 byte cache with 32-byte blocks:
  - The uppermost 22 address bits are the Cache Tag
  - The lowest 5 address bits are the Byte Select (Block Size = 2^5)
  - The next 5 address bits (bit5 - bit9) are the Cache Index
Example: 1K Direct Mapped Cache

<table>
<thead>
<tr>
<th>Cache Index</th>
<th>Cache Tag</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000050</td>
<td>0x0</td>
</tr>
<tr>
<td>1</td>
<td>0x000440</td>
<td>0x1</td>
</tr>
<tr>
<td>2</td>
<td>0x000240</td>
<td>0x2</td>
</tr>
<tr>
<td>3</td>
<td>0x000050</td>
<td>0x3</td>
</tr>
</tbody>
</table>

Valid Bit | Cache Tag | Cache Data
---|-----------|-------------|
0 | 0x000050 | Byte 0 |
1 | 0x000440 | Byte 1 |
2 | 0x000240 | Byte 2 |
3 | 0x000050 | Byte 3 |

Block Size Tradeoff

- In general, larger block size take advantage of spatial locality BUT:
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - If block size is too big relative to cache size, miss rate will go up
    - Too few cache blocks

- In general, Average Access Time:
  - Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate

Advantages of Set associative cache

- Higher Hit rate for the same cache size.
- Fewer Conflict Misses.
- Can can have a larger cache but keep the index smaller (same size as virtual page index)
Disadvantage of Set Associative Cache

° N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
° In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.

Sources of Cache Misses

° Compulsory (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
° Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
° Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
° Invalidation: other process (e.g., I/O) updates memory

Sources of Cache Misses

<table>
<thead>
<tr>
<th>Cache Miss</th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory Miss</td>
<td>Big</td>
<td>Medium</td>
<td>Small</td>
</tr>
<tr>
<td>Conflict Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Capacity Miss</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
</tr>
<tr>
<td>Invalidation Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

Note:
If you are going to run "billion"s of instruction, Compulsory Misses are insignificant.

The Need to Make a Decision!

° Direct Mapped Cache:
  - Each memory location can only map to 1 cache location
  - No need to make any decision
  - Current item replaces the previous item in that cache location
° N-way Set Associative Cache:
  - For each memory location have a choice of N cache locations
° Fully Associative Cache:
  - Each memory location can be placed in ANY cache location
  - Cache miss in a N-way Set Associative or Fully Associative Cache:
    - Bring in new block from memory
    - Throw out a cache block to make room for the new block
    - We need to make a decision on which block to throw out!

Cache Block Replacement Policy

° Random Replacement:
  - Hardware randomly selects a cache item and throw it out
° Least Recently Used:
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time.
  - For two way set associative cache one needs one bit for LRU replacement.
° Example of a Simple "Pseudo" Least Recently Used Implementation:
  - Assume 64 Fully Associative Entries
  - Hardware replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
Cache Write Policy: Write Through versus Write Back

- Cache read is much easier to handle than cache write.
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
  - Need a "dirty bit" for each cache block
  - Greatly reduce the memory bandwidth requirement
  - Control can be complex
  - Write Through: write to cache and memory at the same time.
  - What?! How can this be? Isn't memory too slow for this?

Write Buffer for Write Through

- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if Store frequency (w.r.t. time) << 1 / DRAM write cycle
  - Memory system designer’s nightmare:
    - Store frequency (w.r.t. time) > 1 / DRAM write cycle
    - Write buffer saturation

Write Buffer Saturation

- Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time << DRAM Write Cycle Time
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache:
  - Install a write buffer after the cache
  - Use compression

Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we read in the block?
    - Yes: Write Allocate
    - No: Write Not Allocate

Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? (Block placement)
- Q2: How is a block found if it is in the upper level? (Block identification)
- Q3: Which block should be replaced on a miss? (Block replacement)
- Q4: What happens on a write? (Write strategy)

What is a Sub-block?

- Sub-block:
  - Share one cache tag between all sub-blocks in a block
  - A unit within a block that has its own valid bit
  - Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
    - Each cache entry will have: 32/8 = 4 valid bits
- Write miss: only the bytes in that sub-block is brought in.
  - reduce cache fill bandwidth (penalty).
Summary

° Caches provide cost effective memory system
° Work by exploiting locality (temporal & spatial)
° Associativity, Blocksize, Capacity (ABCs of caches)
° Know how a cache works
  • Break address into tag, index, block offset
° Know how to draw a block diagram of a cache

Next Time
° Cache Performance and Programming