CS152
Computer Architecture and Engineering
Lecture 20

Caches
The Five Classic Components of a Computer

Today’s Topics:
- Recap last lecture
- Simple caching techniques
- Many ways to improve cache performance
- Virtual memory?
Optimize the memory system organization to minimize the average memory access time for typical workloads.

Workload or Benchmark programs

Processor

reference stream
<op,addr>, <op,addr>, <op,addr>, <op,addr>, ...

op: i-fetch, read, write

Memory
$ MEM
Example: 1 KB Direct Mapped Cache with 32 B Blocks

° For a $2^N$ byte cache:
  • The uppermost $(32 - N)$ bits are always the Cache Tag
  • The lowest $M$ bits are the Byte Select (Block Size = $2^M$)
  • One cache miss, pull in complete “Cache Block” (or “Cache Line”)

```
+----------------+----------------+----------------+----------------+
| Cache Tag      | Example: 0x50  | Cache Index    | Byte Select    |
+----------------+----------------+----------------+----------------+
| Ex: 0x01       | Ex: 0x00       |                |                |
+----------------+----------------+----------------+----------------+
| Stored as part of the cache “state” |
+----------------+----------------+----------------+----------------+
| Valid Bit      | Cache Tag      | Cache Data     |                |
|                | 0x50           | Byte 31        |                |
|                |                | **              |                |
|                |                | Byte 63        |                |
|                |                | **              |                |
|                |                | Byte 1023      |                |
|                |                | **              |                |
+----------------+----------------+----------------+----------------+
| Byte 1         | Byte 33        |                |                |
| **              | Byte 32        | **              |                |
| Byte 992       |                |                |                |
|                |                |                |                |
+----------------+----------------+----------------+----------------+```
Set Associative Cache

° **N-way set associative**: N entries for each Cache Index
  - N direct mapped caches operates in parallel

° **Example: Two-way set associative cache**
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result
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.
Example: Fully Associative

**Fully Associative Cache**
- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- Example: Block Size = 32 B blocks, we need N 27-bit comparators

° By definition: Conflict Miss = 0 for a fully associative cache
A Summary on 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
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant

° **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

° **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

° **Coherence** (Invalidation): other process (e.g., I/O) updates memory
## Design options at constant cost

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

**Note:**

If you are going to run “billions” of instruction, Compulsory Misses are insignificant (except for streaming media types of programs).
Recap: Four Questions for Caches and Memory Hierarchy

° 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)*
Q1: Where can a block be placed in the upper level?

° Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

- Fully associative: block 12 can go anywhere
- Direct mapped: block 12 can go only into block 4 (12 mod 8)
- Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Q2: How is a block found if it is in the upper level?

- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativityshrinks index, expands tag
Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Associativity:</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>
Q4: What happens on a write?

- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.

- **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?

- **Pros and Cons of each?**
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes

- **WT always combined with write buffers so that don’t wait for lower level memory**
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
- Must handle bursts of writes
- Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
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: (does this always work?)
Write-Buffer Issues: Could introduce RAW Hazard with memory!
- Write buffer may contain *only* copy of valid data \(\Rightarrow\) Reads to memory may get wrong result if we ignore write buffer

**Solutions:**
- Simply wait for write buffer to empty before servicing reads:
  - Might increase read miss penalty (old MIPS 1000 by 50%)
- Check write buffer contents before read ("fully associative");
  - If no conflicts, let the memory access continue
  - Else grab data from buffer

Can Write Buffer help with Write Back?
- Read miss replacing dirty block
  - Copy dirty block to write buffer *while* starting read to memory
- CPU stall less since restarts as soon as do read
Write-miss Policy: Write Allocate versus Not Allocate

° Assume: a 16-bit write to memory location 0x0 and causes a miss
  • Do we allocate space in cache and possibly read in the block?
    - Yes: Write Allocate
    - No: Not Write Allocate
Impact of Memory Hierarchy on Algorithms

° Today CPU time is a function of (ops, cache misses)

° What does this mean to Compilers, Data structures, Algorithms?
  • Quicksort: fastest comparison based sorting algorithm when keys fit in memory
  • Radix sort: also called “linear time” sort
    For keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys

  • For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000
Quicksort vs. Radix as vary number keys: Instructions

Radix sort

Quick sort

Instructions/key

Job size in keys

Lec20.20
Quicksort vs. Radix as vary number keys: Instrs & Time

![Graph showing comparison between Quicksort and Radix sort for varying numbers of keys. The x-axis represents job size in keys, ranging from 1000 to 1E+7. The y-axis shows time and instructions. The graph compares time and instructions for Quicksort and Radix sort.]
Quicksort vs. Radix as vary number keys: Cache misses

What is proper approach to fast algorithms?
Summary #1/ 2:

° 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 (+1) 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
  • Coherence Misses: Caused by external processors or I/O devices

° Cache Design Space
  • total size, block size, associativity
  • replacement policy
  • write-hit policy (write-through, write-back)
  • write-miss policy
Several interacting dimensions

- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back
- write allocation

The optimal choice is a compromise

- depends on access characteristics
  - workload
  - use (I-cache, D-cache, TLB)
- depends on technology / cost

Simplicity often wins