Lecture 13: Caches

- Today
  - Caches

Memory System Overview

- Memory Hierarchies
  - Latency/Bandwidth/Locality
  - Caches
    - Principles - why does it work
    - Cache organization
    - Cache performance
    - Types of misses (the 3 Cs)
  - Main memory organization
    - DRAM vs. SRAM
    - Bank organization
    - Tracking multiple references
  - Trends in memory system design

- Logical Organization
  - Name spaces
  - Protection and sharing
  - Resource management
    - virtual memory, paging, and swapping
  - Segmentation
  - Capability-based addressing
The Memory Bottleneck

- Typical CPU clock rate
  - 1 GHz (1 ns cycle time)
- Typical DRAM access time
  - 30 ns (about 30 cycles)
- Typical main memory access
  - 100 ns (100 cycles)
    - DRAM (30), precharge (10), chip crossings (30), overhead (30).
- Our pipeline designs assume 1 cycle access (1 ns)
- Average instruction references
  - 1 instruction word
  - 0.3 data words
- This problem gets worse
  - CPUs get faster
  - Memories get bigger
- Memory delay is mostly communication time
  - reading/writing a bit is fast
  - it takes time to
    - select the right bit
    - route the data to/from the bit
- Big memories are slow
- Small memories can be made fast

Cache Memory

- Small fast memory + big slow memory
- Looks like a big fast memory
The Memory Hierarchy

Latency | Bandwidth | Managed By
--- | --- | ---
1 cyc | 3-10 words/cycle < 1KB | compiler managed
1-3cy | 1-2 words/cycle 32KB - 1MB | hardware managed
5-10cy | 1 word/cycle 1MB - 4MB | hardware managed
30-100cy | 0.5 words/cycle 64MB - 4GB | OS managed
10⁵-10⁷cy | 0.01 words/cycle 4GB+ | OS managed

Where Does the Memory Hierarchy Fit In?
Typical Cache Organization

Locality of Reference

- **Spatial Locality**
  - likely to reference data near recent references

- **Temporal Locality**
  - likely to reference the same data that was referenced recently
Program Behavior

- Locality depends on type of program
- Some programs 'behave' well
  - small loop operating on data on stack
- Some programs don’t
  - frequent calls to nearly random subroutines
  - traversal of large, sparse data set
    - essentially random data references with no reuse
- Most programs exhibit some degree of locality

Example

What is the average memory access time?

\[ AMAT = \text{Latency}_{\text{Hit}} + P(\text{miss}) \times \text{Latency}_{\text{Miss}} \]
Impact of Hit Rate

Average Access Time

Hit Rate

Cache Organization

Address

Valid bits

Data

<table>
<thead>
<tr>
<th>Address</th>
<th>27</th>
<th>15</th>
<th>42</th>
<th>86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>95</td>
<td>11</td>
<td>75</td>
<td>33</td>
</tr>
<tr>
<td>bits</td>
<td>90</td>
<td>12</td>
<td>74</td>
<td>35</td>
</tr>
<tr>
<td>Data</td>
<td>99</td>
<td>13</td>
<td>73</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>14</td>
<td>72</td>
<td>37</td>
</tr>
</tbody>
</table>

- Where does a block get placed?
- How do we find it?
- Which one do we replace when a new one is brought in?
- What happens on a write?
Cache Definitions

- Cache block (= cache line)
- Miss rate
- Miss penalty
- Index
- Tag
- Offset

Where Does a Block Go in the Cache?

- Where do we put block 12?
  - Word = 4 bytes
  - Block = 1 word
Direct Mapped

- Each block mapped to exactly 1 cache location

  Cache location = (block address) MOD (# blocks in cache)

![Diagram of Direct Mapped Cache]

Fully Associative

- Each block mapped to any cache location

  Cache location = any

![Diagram of Fully Associative Cache]
Set Associative

- Each block mapped to subset of cache locations

  Set selection = (block address) MOD (# sets in cache)

  ![Diagram of set associative cache](image)

  - 2-way set associative = 2 blocks in set
  - This example: 4 sets

Block Placement

- Mapping function from Big Memory to Small memory
- On block-by-block basis
  - Direct Mapped: 1 place
  - Fully Associative: Anywhere
  - Set Associative: Subset of cache
- Use address to do mapping and lookup
Taking advantage of Spatial Locality

- Instead of each block in cache being just 1 word, what if we made it 4 words?
- When we get our 1 word instruction or 1 word of data from memory to put in the cache, get the next 3 as well, because they are likely to be used soon!
- Need to add a way to choose which of the 4 words in the block we want when we go to cache... called block offset.

How Do We Find a Block in The Cache?

- Our Example:
  - Main memory address space = 32 bits (= 4GBytes)
  - Block size = 4 words = 16 bytes
  - Cache capacity = 8 blocks = 128 bytes

- Valid bit \(\Rightarrow\) is cache block good?
- index \(\Rightarrow\) which set
- tag \(\Rightarrow\) which data/instruction in block
- block offset \(\Rightarrow\) which word in block
- # tag/index bits determine the associativity
- tag/index bits can come from anywhere in block address
Finding a Block: Direct-Mapped

Finding A Block: 2-Way Set-Associative
Set Associative Cache

- \( S \) - sets
- \( A \) - elements in each set
  - \( A \)-way associative
- In the example, \( S=4, A=2 \)
  - 4-way associative 8-entry cache
- All of main memory is divided into \( S \) sets
  - All addresses in set \( N \) map to same set of the cache
    - \( \text{Addr} = N \mod S \)
    - \( A \) locations available
- Shares costly comparators across sets
- Low address bits select set
  - 2 in example
- High address bits are \textit{tag}, used to associatively search the selected set
- Extreme cases
  - \( A=1 \): Direct mapped cache
  - \( S=1 \): Fully associative
- \( A \) need not be a power of 2

Finding A Block: Fully Associative
Summary

- Cache overview and organization

- Next Time
  - Cache misses and optimization