Cache Memories

CMPU 224 – Computer Organization
Jason Waterman
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

![Graph showing the gap between DRAM, disk, and CPU speeds over time.](image)

- Disk seek time
- SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time

11/20/2019

CMPU 224 -- Computer Organization
Cache Memories

• **Cache memories** are small, fast SRAM-based memories managed automatically in hardware
  - Hold frequently accessed blocks of main memory
• CPU looks first for data in cache, then in main memory
• Typical system structure:
But the cache is small, why does it help?

• Locality to the rescue!
  • The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

• Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

• Temporal locality:
  • Recently referenced items are likely to be referenced again in the near future

• Spatial locality:
  • Items with nearby addresses tend to be referenced close together in time
Cache Concepts

• Example:
  • 1024 byte main memory
  • 10 bit addresses
  • 64 byte cache

• Once we place memory in the cache how do we find it?

• Need to store some additional identification information along with the data
Cache Concepts: Blocks

- Example:
  - 1024 byte main memory
  - 10 bit addresses
  - 64 byte cache
  - 4 byte blocks

- Once we place memory in the cache how do we find it?

- Organize data into blocks
  - Data will be brought into the cache in block sized chunks
Cache Concepts: Finding a Block

- **Example**: get the byte of data for memory address: \(0x015\) (0000010101)

- First search cache before looking in main memory
  - What block to look for?

- Take address and divide into block and offset bits
  - Block: 5
  - Offset: 1

- Search each of the blocks in the cache for block 5
  - Match! (aka cache hit)

- Get the byte of data at offset 1 for \(0x27688243\)
General Cache Concepts

Larger, slower, cheaper memory viewed as partitioned into “blocks”.

Data is copied in block-sized transfer units.

Smaller, faster, more expensive memory caches a subset of the blocks.
General Cache Concepts: Hit

Data in block 14 is needed

Block 14 is in cache: Hit!
General Cache Concepts: Miss

Data in block 12 is needed

Block 12 is not in cache:
Miss!

Block 12 is fetched from memory

Block 12 is stored in cache
• Replacement policy: determines which block gets evicted (victim)
Cache lookups

• We want to see if our block in the cache as quickly as possible
• It may take too long to search every block in the cache for larger caches
  • To speed lookups, restrict where a block can reside in the cache
    • Divide our cache into sets
    • Each block is mapped to a single set based on its address

```
tag  set  block offset
0110010101
```

Cache

- Set 0
- Set 1
- Set 2
- Set 3
General Cache Organization (S, E, B)

\[ S = \log_2(S) \]

\[ E = 2^e \text{ lines per set} \]

\[ S = 2^s \text{ sets} \]

\[ B = 2^b \text{ bytes per cache block (the data)} \]

**Cache size:**

\[ C = S \times E \times B \text{ data bytes} \]
Cache Read

- $E = 2^s$ lines per set
- $S = 2^s$ sets

1. Locate set
2. Check if any line in set has matching tag
3. Yes + line valid: hit
4. Locate data starting at offset

Address of word:
- $t$ bits for tag
- $s$ bits for set index
- $b$ bits for block offset

Data begins at this offset

Valid bit

$B = 2^b$ bytes per cache block (the data)
Example: Read an int from a Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

\[ S = 2^s \text{ sets} \]

Address of int: 
\[ \text{t bits} \quad 0...01 \quad 100 \]

find set
Example: Read an int from a Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

Address of int:

t bits
0...01 100

match: yes = hit
valid?

block offset
Example: Read an int from a Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

If tag doesn’t match: old line is evicted and replaced
Direct-Mapped Cache Simulation

M=16 bytes (4-bit addresses), B=2 bytes/block, S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0000&lt;sub&gt;2&lt;/sub&gt;]</td>
<td>miss</td>
</tr>
<tr>
<td>1</td>
<td>[0001&lt;sub&gt;2&lt;/sub&gt;]</td>
<td>hit</td>
</tr>
<tr>
<td>7</td>
<td>[0111&lt;sub&gt;2&lt;/sub&gt;]</td>
<td>miss</td>
</tr>
<tr>
<td>8</td>
<td>[1000&lt;sub&gt;2&lt;/sub&gt;]</td>
<td>miss</td>
</tr>
<tr>
<td>0</td>
<td>[0000&lt;sub&gt;2&lt;/sub&gt;]</td>
<td>miss</td>
</tr>
</tbody>
</table>

Set 0:
- v = 1, Tag = 0, Block = M[0-1]
- v = 0
- v = 0
- v = 1, Tag = 0, Block = M[6-7]
2-way Set Associative Cache

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of short integer:

| t bits | 0...01 | 100 |

Find set
2-way Set Associative Cache

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of short integer:

```
| t bits | 0...01 | 100 |
```

valid? + match: yes = hit

block offset
2-way Set Associative Cache

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of short integer:

<table>
<thead>
<tr>
<th>t bits</th>
<th>0...01</th>
<th>100</th>
</tr>
</thead>
</table>

valid? + match: yes = hit

short integer (2 Bytes) is here

No match:
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...
**2-Way Set Associative Cache Simulation**

\[ t=2 \quad s=1 \quad b=1 \]

- \( M = 16 \) byte addresses, \( B = 2 \) bytes/block,
- \( S = 2 \) sets, \( E = 2 \) blocks/set

Address trace (reads, one byte per read):

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>( \text{M}[0-1] )</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>( \text{M}[6-7] )</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>( \text{M}[8-9] )</td>
</tr>
</tbody>
</table>

\[ xx \quad x \quad x \]
The following table gives the parameters for a number of different caches, where $m$ is the number of physical address bits, $C$ is the cache size (number of data bytes), $B$ is the block size in bytes, and $E$ is the number of lines per set. For each cache, determine the number of cache sets ($S$), tag bits ($t$), set index bits ($s$), and block offset bits ($b$).

<table>
<thead>
<tr>
<th>$m$</th>
<th>$C$</th>
<th>$B$</th>
<th>$E$</th>
<th>$S$</th>
<th>$t$</th>
<th>$s$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>1024</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1024</td>
<td>4</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1024</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1024</td>
<td>8</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1024</td>
<td>32</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1024</td>
<td>32</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Lookup Practice

- Memory is byte addressable
- Addresses are 12 bits wide, cache is 4-way set associative, with a 4 byte block size, and 8 total lines
- Address: 0xE34
  - Fill out the tables below

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte0</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_0</td>
<td>058</td>
<td>0</td>
<td>02</td>
<td>55</td>
<td>AD</td>
<td>87</td>
</tr>
<tr>
<td>S_0</td>
<td>123</td>
<td>1</td>
<td>3E</td>
<td>98</td>
<td>47</td>
<td>51</td>
</tr>
<tr>
<td>S_0</td>
<td>12B</td>
<td>0</td>
<td>6C</td>
<td>77</td>
<td>89</td>
<td>14</td>
</tr>
<tr>
<td>S_0</td>
<td>0EF</td>
<td>1</td>
<td>B9</td>
<td>64</td>
<td>78</td>
<td>25</td>
</tr>
<tr>
<td>S_1</td>
<td>069</td>
<td>0</td>
<td>00</td>
<td>FF</td>
<td>14</td>
<td>43</td>
</tr>
<tr>
<td>S_1</td>
<td>12B</td>
<td>1</td>
<td>92</td>
<td>63</td>
<td>42</td>
<td>21</td>
</tr>
<tr>
<td>S_1</td>
<td>075</td>
<td>0</td>
<td>33</td>
<td>BE</td>
<td>AF</td>
<td>31</td>
</tr>
<tr>
<td>S_1</td>
<td>1C6</td>
<td>1</td>
<td>22</td>
<td>17</td>
<td>02</td>
<td>24</td>
</tr>
</tbody>
</table>

Parameter | Value
---|---
Block offset
Set Index
Cache Tag
Cache Hit? (Y/N)
Cache Byte returned
Cache Lookup Practice

- Memory is byte addressable
- Addresses are 12 bits wide, cache is 4-way set associative, with a 4 byte block size, and 8 total lines
- Address: 0x95B
  - Fill out the tables below
Intel Core i7 Cache Hierarchy

Processor package

Core 0
- Regs
- L1 d-cache
- L1 i-cache
- L2 unified cache
- L3 unified cache (shared by all cores)

Core 3
- Regs
- L1 d-cache
- L1 i-cache
- L2 unified cache

L1 i-cache and d-cache:
- 32 KB, 8-way,
- Access: 4 cycles

L2 unified cache:
- 256 KB, 8-way,
- Access: 10 cycles

L3 unified cache:
- 8 MB, 16-way,
- Access: 40-75 cycles

Block size: 64 bytes for all caches.

Main memory