Caches III
CSE 351 Spring 2020

Instructor:
Ruth Anderson

Teaching Assistants:
Alex Olshanskyy
Rehaan Bhimani
Callum Walker
Chin Yeoh
Diya Joy
Eric Fan
Edan Sneh
Jonathan Chen
Jeffery Tian
Millicent Li
Melissa Birchfield
Porter Jones
Joseph Schafer
Connie Wang
Eddy (Tianyi) Zhou

https://what-if.xkcd.com/111/
Administrivia

- Unit Summary #2 due TONIGHT, Friday (5/08)
  - Submit to Canvas – TWO SEPARATE SUBMISSIONS
- Lab 3 due Wednesday (5/13)

- You must log on with your @uw google account to access!!
  - Google doc for 11:30 Lecture: [https://tinyurl.com/351-05-08A](https://tinyurl.com/351-05-08A)
  - Google doc for 2:30 Lecture: [https://tinyurl.com/351-05-08B](https://tinyurl.com/351-05-08B)
Making memory accesses fast!

- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (sets; index + tag)
  - Associativity (ways)
  - Replacement policy
  - Handling writes
- Program optimizations that consider caches
Review: Direct-Mapped Cache

Hash function: \((\text{block number}) \mod (\# \text{ of blocks in cache})\)
- Each memory address maps to \textit{exactly} one index in the cache
- Fast (and simpler) to find a block

Here \(K = 4\ B\) and \(C/K = 4\)
Direct-Mapped Cache Problem

What happens if we access the following addresses?

- 8, 24, 8, 24, 8, ...?
  - Conflict in cache (misses!)
  - Rest of cache goes unused

Solution?

Memory

<table>
<thead>
<tr>
<th>Block Num</th>
<th>Block Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Block Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>CATZ</td>
</tr>
<tr>
<td>11</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>

Here $K = 4$ B and $C/K = 4$
Associativity

- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we combine the two ideas:
  - Each address maps to exactly one set
  - Each set can store block in more than one way
Cache Organization (3)

- **Associativity** \((E)\): # of ways for each set
  - Such a cache is called an “\(E\)-way set associative cache”
  - We now index into cache *sets*, of which there are \(S = C/K/E\)
  - Use lowest \(\log_2(C/K/E) = s\) bits of block address
    - **Direct-mapped**: \(E = 1\), so \(s = \log_2(C/K)\) as we saw previously
    - **Fully associative**: \(E = C/K\), so \(s = 0\) bits

Note: The textbook uses “b” for offset bits.
Example Placement

- Where would data from address 0x1833 be placed?
  - Binary: \(0b \ 0001 \ 1000 \ 0011 \ 0011\)

\[
t = m - s - k \quad s = \log_2(C/K/E) \quad k = \log_2(K)
\]

\(m\)-bit address: Tag \(t\) | Index \(s\) | Offset \(k\)

- Direct-mapped
- 2-way set associative
- 4-way set associative
Block Replacement

- Any empty block in the correct set may be used to store block.
- If there are no empty blocks, which one should we replace?
  - No choice for direct-mapped caches.
  - Caches typically use something close to least recently used (LRU)
    (hardware usually implements “not most recently used”)

<table>
<thead>
<tr>
<th>Direct-mapped</th>
<th>2-way set associative</th>
<th>4-way set associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Tag</td>
<td>Data</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Valid} = 0 \]
Polling Question [Cache III]

- We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?
  - Vote at [http://pollev.com/rea](http://pollev.com/rea)
  - A. 2
  - B. 4
  - C. 8
  - Each set has 8 blocks, so $E = 8$
  - D. 16
  - E. We’re lost...

- If addresses are 16 bits wide, how wide is the Tag field? $k = \log_2(K) = 7$ bits, $s = \log_2(S) = 1$ bit, $t = m - s - k = 8$ bits
**General Cache Organization \((S, E, K)\)**

- \(E = \) blocks (or lines) per set
- \(S = \) sets
- \(S = 2^s\)
- \(K = \) bytes per block
- Cache size:
  \[ C = K \times E \times S \text{ data bytes} \]
  \(\text{(doesn’t include } V \text{ or Tag)}\)
Notation Review

- We just introduced a lot of new variable names!
  - Please be mindful of block size notation when you look at past exam questions or are watching videos

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>$K \ (B \text{ in book})$</td>
<td></td>
</tr>
<tr>
<td>Cache size</td>
<td>$C$</td>
<td>$M = 2^m$ $\leftrightarrow m = \log_2 M$</td>
</tr>
<tr>
<td>Associativity</td>
<td>$E$</td>
<td>$S = 2^s$ $\leftrightarrow s = \log_2 S$</td>
</tr>
<tr>
<td>Number of Sets</td>
<td>$S$</td>
<td>$K = 2^k$ $\leftrightarrow k = \log_2 K$</td>
</tr>
<tr>
<td>Address space</td>
<td>$M$</td>
<td>$C = K \times E \times S$</td>
</tr>
<tr>
<td>Address width</td>
<td>$m$</td>
<td>$s = \log_2 \left(\frac{C}{K/E}\right)$</td>
</tr>
<tr>
<td>Tag field width</td>
<td>$t$</td>
<td>$m = t + s + k$</td>
</tr>
<tr>
<td>Index field width</td>
<td>$s$</td>
<td></td>
</tr>
<tr>
<td>Offset field width</td>
<td>$k \ (b \text{ in book})$</td>
<td></td>
</tr>
</tbody>
</table>
Example Cache Parameters Problem

- 4 KiB address space, 125 cycles to go to memory.

Fill in the following table:

<table>
<thead>
<tr>
<th>Cache Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>256 B</td>
</tr>
<tr>
<td>Block Size</td>
<td>32 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>2-way</td>
</tr>
<tr>
<td>Hit Time</td>
<td>3 cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>20%</td>
</tr>
<tr>
<td>Tag Bits</td>
<td>5</td>
</tr>
<tr>
<td>Index Bits</td>
<td>2</td>
</tr>
<tr>
<td>Offset Bits</td>
<td>5</td>
</tr>
<tr>
<td>AMAT</td>
<td>3 + 0.2(125)  = 28 clock cycles</td>
</tr>
</tbody>
</table>

Equations:
- \( 2^k \times 8 \implies m = 12 \text{ bits} \) (MP)
- \( t = m - s - k \)
- \( s = \log_2(C/K/E) \)
- \( k = \log_2(K) \)
- \( \text{AMAT} = \text{HT} + \text{MR} \times \text{MP} \)
Cache Read

1) Locate set
2) Check if any line in set is valid and has matching tag: (hit)
3) Locate data starting at offset

Address of byte in memory:
- \( t \) bits
- \( s \) bits
- \( k \) bits

- tag
- set index
- block offset

data begins at this offset

valid bit

\( S = \# \text{ sets} = 2^s \)

\( E = \text{blocks/lines per set} \)

\( K = \text{bytes per block} \)
Example: Direct-Mapped Cache ($E = 1$)

Direct-mapped: One line per set
Block Size $K = 8$ B

$S = 2^s$ sets

Address of `int`:

```
0...01 100
```

find set

```
\begin{array}{|c|c|c|c|c|c|c|}
\hline
V & Tag & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
V & Tag & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
V & Tag & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\vdots
\hline
V & Tag & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\end{array}
```
Example: Direct-Mapped Cache ($E = 1$)

Direct-mapped: One line per set
Block Size $K = 8$ B
Example: Direct-Mapped Cache \((E = 1)\)

Direct-mapped: One line per set
Block Size \(K = 8\) B

No match? Then old line gets evicted and replaced

This is why we want alignment!

No unnecessary extra cache accesses across block boundaries
Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set
Block Size $K = 8$ B

Address of `short int`:

| 8 bits | 0...01 | 100 |

find set

"way 0"

"way 1"
Example: Set-Associative Cache \((E = 2)\)

2-way: Two lines per set
Block Size \(K = 8\) B

Valid? + Match: yes = hit

Address of short int:

Block offset

\(8\) bits \(0\ldots01\ 100\)
Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set
Block Size $K = 8$ B

- valid? + match: yes = hit

Address of short int:
- compare both
- block offset
- short int (2 B) is here

No match?
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...
Types of Cache Misses: 3 C’s!

- **Compulsory** (cold) miss
  - Occurs on first access to a block

- **Conflict** miss
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - *e.g.* referencing blocks 0, 8, 0, 8, ... could miss every time
  - Direct-mapped caches have more conflict misses than $E$-way set-associative (where $E > 1$)

- **Capacity** miss
  - Occurs when the set of active cache blocks (the *working set*) is larger than the cache (just won’t fit, even if cache was fully-associative)
  - **Note:** Fully-associative only has Compulsory and Capacity misses
Example Code Analysis Problem

- Assuming the cache starts **cold** (all blocks invalid) and \( \text{sum}, i, \) and \( j \) are stored in registers, calculate the **miss rate**:
  - \( m = 12 \) bits, \( C = 256 \) B, \( K = 32 \) B, \( E = 2 \)

  \[
  \frac{1}{4}
  \]

```
# define SIZE 8
long ar[SIZE][SIZE], sum = 0; // &ar=0x800
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[i][j];
```

**Challenge:** what is the miss rate if we switch the ordering of the for-loops?