Caches II
CSE 351 Autumn 2016

Instructor:
Justin Hsia

Teaching Assistants:
Chris Ma
Hunter Zahn
John Kaltenbach
Kevin Bi
Sachin Mehta
Suraj Bhat
Thomas Neuman
Waylon Huang
Xi Liu
Yufang Sun

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

- Lab 3 due Thursday

- **Midterm grades**
  - Unadjusted average currently right around 65%
    - All scores will be adjusted up by 6 points in Catalyst
  - Regrade requests open on [Gradescope](https://gradescope.com) until end of Thursday
    - It is possible for your grade to go down
    - Make sure you submit separate requests for each portion of a question (e.g. Q5A and Q5B) – these may go to different graders!

![Midterm Score Distribution](chart.png)
An Example Memory Hierarchy

- CPU registers hold words retrieved from L1 cache
- L1 cache holds cache lines retrieved from L2 cache
- L2 cache holds cache lines retrieved from main memory
- Main memory holds disk blocks retrieved from local disks
- Local disks hold files retrieved from disks on remote network servers
- Remote secondary storage (distributed file systems, web servers)

L1: registers on-chip L1 cache (SRAM)
L2: off-chip L2 cache (SRAM)
Main memory (DRAM)
Local secondary storage (local disks)
Remote secondary storage (distributed file systems, web servers)

Smaller, faster, costlier per byte
Larger, slower, cheaper per byte
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
Cache Organization (1)

- **Block Size ($K$):** unit of transfer between $\$ and Mem
  - Given in bytes and always a power of 2 (e.g. 64 B) \( = 2^6 \iff 6 = \log_2(64) \)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!

Within Block: (Lab 1)

<table>
<thead>
<tr>
<th>Block Number</th>
<th>Block Offset</th>
<th>Which Block?</th>
<th>Where in Block?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0...00</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>0...00</td>
<td>11111111</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0...01</td>
<td>00000000</td>
<td></td>
</tr>
</tbody>
</table>
Cache Organization (1)

- **Block Size** \((K)\): unit of transfer between $\$ and Mem
  - Given in bytes and always a power of 2 (e.g. 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!
- **Offset field**
  - Low-order \(\log_2(K) = 0\) bits of address tell you which byte within a block
  - \((\text{address}) \mod 2^n = n\) lowest bits of address
  - \((\text{address}) \text{ modulo} \ (# \text{ of bytes in a block})\)

\[
\text{A-bit address: } \begin{array}{c|c|c}
\text{A} - 0 \text{ bits} & \text{Block Number} & 0 \text{ bits} \\
\hline
\text{(refers to byte in memory)} & \\
\end{array}
\]
Cache Organization (2)

- **Cache Size** ($C$): amount of *data* the cache can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes ($C$) or number of blocks ($C/K$)
  - *Example*: $C = 32$ KiB = 512 blocks if using 64-B blocks

- Where should data go in the cache?
  - We need a mapping from memory addresses to specific locations in the cache to make checking the cache for an address *fast*

- What is a data structure that provides fast lookup?
  - Hash table!
Aside: Hash Tables for Fast Lookup

Insert: \[ \mod 10 \]

- 5
- 27
- 34
- 102
- 119

Apply hash function to map data to “buckets”
- want fast computation
- want to use all buckets “well”
Place Data in Cache by Hashing Address

Map to *cache index* from block address

- Use next $\log_2(C/K)$ = \( I \) bits = 2
- (block address) mod (# blocks in cache)
- Lets adjacent blocks fit in cache simultaneously!

Here $K = 4$ B and $C/K = 4$
Place Data in Cache by Hashing Address

Collision!
- This might confuse the cache later when we access the data
- Solution?

Here $K = 4$ B and $C/K = 4$
Tags Differentiate Blocks in Same Index

Tag = rest of address bits
- \( T \) bits = \( A - I - O \)
- Check this during a cache lookup

Here \( K = 4 \) B and \( C/K = 4 \)
Checking for a Requested Address

- CPU sends address request for chunk of data
  - Address and requested data are not the same thing!
    - Analogy: your friend ≠ his or her phone number

- TIO address breakdown:
  - A-bit address: [Tag (T)] [Index (I)] [Offset (O)]
  - **Index** field tells you where to look in cache
  - **Tag** field lets you check that data is the block you want
  - **Offset** field selects specified start byte within block

- Note: T and I sizes will change based on hash function
Cache Puzzle #1

- What can you infer from the following behavior?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (14: miss), (15: hit), (16: miss)

  \[ \Rightarrow 14 \text{ and } 15 \text{ are in the same block} \]
  \[ \Rightarrow 15 \text{ and } 16 \text{ are not in the same block} \]
  \[ \Rightarrow \text{pulls block containing } 14 \text{ into } \$
  
- Minimum block size? \[2B\]

- Maximum block size? \[16B\]
Direct-Mapped Cache

Memory

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

Hash function: \((\text{block address}) \mod (\# \text{ of blocks in cache})\)

- Each memory address maps to \textit{exactly} one index in the cache
- Fast (and simpler) to find an address

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?

Here $K = 4$ B and $C/K = 4$
$I = \log_2(C/K) = 2$
$0 = \log_2(k) = 2$
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

![Diagram of cache sets and ways](image)

1-way:
- 8 sets, 1 block each

2-way:
- 4 sets, 2 blocks each

4-way:
- 2 sets, 4 blocks each

8-way:
- 1 set, 8 blocks

Direct mapped

Fully associative
Cache Organization (3)

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

<table>
<thead>
<tr>
<th>Used for tag comparison</th>
<th>Selects the set</th>
<th>Selects the byte from block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag ((T))</td>
<td>Index ((I))</td>
<td>Offset ((O))</td>
</tr>
</tbody>
</table>

- Decreasing associativity
  - Direct mapped (only one way)
- Increasing associativity
  - Fully associative (only one set)
Example Placement

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

\[
T = A - I - O \quad I = \log_2 (C/K/N) \quad O = \log_2 (K) = 4
\]

A-bit address:

<table>
<thead>
<tr>
<th>Tag (T)</th>
<th>Index (I)</th>
<th>Offset (O)</th>
</tr>
</thead>
</table>

- \( T = 1 \)  
- \( I = ? \)  
- \( O = ? \)

- Direct-mapped (N=1)
- 2-way set associative (N=2)
- 4-way set associative (N=4)
Example Placement

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

<table>
<thead>
<tr>
<th>Block size:</th>
<th>16 B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity:</td>
<td>8 blocks</td>
</tr>
<tr>
<td>Address:</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

$T = \text{A} - \text{I} - \text{O}$  
$I = \log_2(C/K/N)$  
$O = \log_2(K)$

A-bit address:

<table>
<thead>
<tr>
<th>Tag (T)</th>
<th>Index (I)</th>
<th>Offset (O)</th>
</tr>
</thead>
</table>

$I = 3$

Direct-mapped

$I = 2$

2-way set associative

$I = 1$

4-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<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>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<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>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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”)

---

**Direct-mapped**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<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>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**2-way set associative**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**4-way set associative**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Puzzle #2

- What can you infer from the following behavior?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (10: miss), (12: miss), (10: miss)

  - 12’s block evicted 10’s block
  - 10 and 12 are not in the same block
  - Pulls block containing 10 into $\$

- Associativity?
  - $N = 1$ (direct-mapped)
  - If $N > 1$, both blocks could coexist in same set

- Number of sets?
  - $S \leq 2$
  - Blocks are contiguous, so since 10 and 12 are only two addresses apart, either $C/K = 1$ (1 set means always replace), or 2 sets with $K = 1B$. 
General Cache Organization \((S, N, K)\)

- \(N = \text{blocks/lines per set}\)
- \(S = \text{# sets} = 2^I\)
- \(K = \text{bytes per block}\)
- \(C = S \times N \times K\) data bytes (doesn’t include \(V\) or Tag)
Notation Changes

- We are using different variable names from previous quarters
  - Please be aware of this when you look at past exam questions or are watching videos

<table>
<thead>
<tr>
<th>Variable</th>
<th>This Quarter</th>
<th>Previous Quarters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>K</td>
<td>B</td>
</tr>
<tr>
<td>Cache size</td>
<td>C</td>
<td>---</td>
</tr>
<tr>
<td>Associativity</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>Address width</td>
<td>A</td>
<td>m</td>
</tr>
<tr>
<td>Tag field width</td>
<td>T</td>
<td>t</td>
</tr>
<tr>
<td>Index field width</td>
<td>I</td>
<td>k, s</td>
</tr>
<tr>
<td>Offset field width</td>
<td>O</td>
<td>n, b</td>
</tr>
<tr>
<td>Number of Sets</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>