Caches II
CSE 410 Winter 2017

Instructor: Justin Hsia
Teaching Assistants: Kathryn Chan, Kevin Bi, Ryan Wong, Waylon Huang, Xinyu Sui
Administrivia

- Lab 3 due Thursday (2/23)
- Homework 4 released today (Structs, Caches)

**Mid-Quarter Survey Feedback**

- Pace is “moderate” to “a bit too fast” and course is more than 3 units of work
- You talk too fast in lecture (or rush at the end) and I wish there were more peer instruction questions
- Canvas quiz answer keys are annoying, but instant homework feedback is great
- Sections: “shorten discussions and lengthen explanations” 😞
Review: Example Memory Hierarchy

- **registers**
- **on-chip L1 cache (SRAM)**
- **off-chip L2 cache (SRAM)**
- **main memory (DRAM)**
- **local secondary storage (local disks)**
- **remote secondary storage (distributed file systems, web servers)**

**Start to bottom:**
- Larger, slower, cheaper per byte
- Smaller, faster, costlier per byte

**Arrow direction:**
- \(\text{on-chip L1 cache (SRAM)}\) holds cache lines retrieved from \(\text{off-chip L2 cache (SRAM)}\)
- \(\text{L1 cache holds cache lines retrieved from L2 cache}\)
- \(\text{L2 cache holds cache lines retrieved from main memory}\)
- \(\text{Main memory holds disk blocks retrieved from local disks}\)
- \(\text{Local disks hold files retrieved from disks on remote network servers}\)
- CPU registers hold words retrieved from L1 cache
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 $S$ 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!

*Note: The textbook uses “B” for block size*
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 \( \log_2(K) = k \) bits of address tell you which byte within a block
    - \((\text{address}) \mod 2^n = n \) lowest bits of address
  - \((\text{address}) \mod \) (number of bytes in a block)

\[m\text{-bit address: } \begin{array}{c|c|c}
  \text{m - k bits} & \text{k bits} \\
  \hline
  \text{Block Number} & \text{Block Offset}
\end{array}\]

(Refers to byte in memory)

**Note:** The textbook uses “b” for offset bits
Cache Organization (2)

- **Cache Size** \( C \): amount of *data* the $ can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes \( C \) or number of blocks \( C/K \)
  - Example: \( C = 32 \text{ KiB} = 512 \) 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!
Review: Hash Tables for Fast Lookup

Insert:
- 5
- 27
- 34
- 102
- 119

Apply hash function to map data to “buckets”
Place Data in Cache by Hashing Address

- Map to cache index from block address
  - Use next $\log_2(C/K) = s$ bits
  - (block address) mod (# blocks in cache)

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

- Map to cache index from block address
- Lets adjacent blocks fit in cache simultaneously!
  - Consecutive blocks go in consecutive cache indices

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 = $m - s - k$
  - 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:
  - $m$-bit address: $\begin{array}{c}
  \text{Tag (t)} \\
  \text{Index (s)} \\
  \text{Offset (k)} \\
  \end{array}$
  - **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 $s$ sizes will change based on hash function
Cache Puzzle #1

Based on the following behavior, which of the following block sizes is NOT possible for our cache?
- Cache starts *empty*, also known as a *cold cache*
- Access (addr: hit/miss) stream:
  - (14: miss), (15: hit), (16: miss)

A. 4 bytes
B. 8 bytes
C. 16 bytes
D. 32 bytes
E. We’re lost...

**Direct-Mapped Cache**

### Hash function:
- A hash function maps each memory address to exactly one index in the cache.
- The function is (block address) mod (number of blocks in the cache).
- This method is fast and simpler to find an address.

### Example:
- Consider a cache with 4 blocks and 16 memory addresses:
  - Hash function: \[ \text{address} \mod 4 \]
  - Mapping:
    - Memory addresses: 00 00, 00 01, 00 10, 00 11, ..., 11 11
    - Cache indices: 00, 01, 10, 11

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?
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

- 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
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 \( C / K / E \)
  - 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:

<table>
<thead>
<tr>
<th>Tag ($t$)</th>
<th>Index ($s$)</th>
<th>Offset ($k$)</th>
</tr>
</thead>
</table>

$s = ?$

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>

$s = ?$

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>

$s = ?$

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>

block size: 16 B
capacity: 8 blocks
address: 16 bits
Example Placement

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

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

- Block size: 16 B
- Capacity: 8 blocks
- Address: 16 bits

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

- \(s = 3\)
  - Direct-mapped

- \(s = 2\)
  - 2-way set associative

- \(s = 1\)
  - 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”)

[Diagrams showing direct-mapped, 2-way set associative, and 4-way set associative cache configurations]
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)

- Associativity?

- Number of sets?
General Cache Organization \((S, E, K)\)

- \(E\) = blocks/lines per set
- \(S\) = \# sets = \(2^s\)
- \(K\) = bytes per block
- Cache size:
  \[ C = K \times E \times S \text{ data bytes} \]
  (doesn’t include \(V\) 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>Variable</th>
<th>This Quarter</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>$K$ ($B$ in book)</td>
<td>$M = 2^m \leftrightarrow m = \log_2 M$</td>
</tr>
<tr>
<td>Cache size</td>
<td>$C$</td>
<td>$S = 2^s \leftrightarrow s = \log_2 S$</td>
</tr>
<tr>
<td>Associativity</td>
<td>$E$</td>
<td>$K = 2^k \leftrightarrow k = \log_2 K$</td>
</tr>
<tr>
<td>Number of Sets</td>
<td>$S$</td>
<td>$C = K \times E \times S$</td>
</tr>
<tr>
<td>Address space</td>
<td>$M$</td>
<td>$s = \log_2 (C/K/E)$</td>
</tr>
<tr>
<td>Address width</td>
<td>$m$</td>
<td>$m = t + s + k$</td>
</tr>
<tr>
<td>Tag field width</td>
<td>$t$</td>
<td></td>
</tr>
<tr>
<td>Index field width</td>
<td>$s$</td>
<td></td>
</tr>
<tr>
<td>Offset field width</td>
<td>$k$ ($b$ in book)</td>
<td></td>
</tr>
</tbody>
</table>