IBM’s hub for wearables could shorten your hospital stay

Researchers at IBM have developed a hub for wearables that can gather information from multiple wearable devices.

The gadget, [called ‘Chiyo’], funnels data from devices such as smart watches and fitness bands into the IBM Cloud. There, it's analyzed and the results are shared with the user and their doctor.

IBM isn't planning to get into the wearables business. Instead, it plans to offer the service as a platform on which other companies can build their own health services.

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

- **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** (DRAM) holds disk blocks retrieved from local disks.
- **Local secondary storage** (local disks) holds files retrieved from disks on remote network servers.
- **Remote secondary storage** (distributed file systems, web servers) is larger, slower, and cheaper per byte.

The hierarchy is structured as a pyramid, with the smallest and fastest components at the top (CPU registers) and the largest and slowest components at the bottom (remote secondary storage). Each level is smaller, faster, and costlier per byte compared to the next level below.
Making memory accesses fast!

- Cache basics
- Principle of locality
- Memory hierarchies
- **Cache organization**
  - Direct-mapped \((sets; \text{index} + \text{tag})\)
  - 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)
  - 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
    - $(address) \mod 2^n = n$ lowest bits of address
    - $(address)$ modulo (# of bytes in a block)

Note: The textbook uses “b” for offset bits
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!
Review: Hash Tables for Fast Lookup

Apply hash function to map data to “buckets”

Goals:
1. fast/simple calculation
2. use all buckets “well”
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

Memory

<table>
<thead>
<tr>
<th>Block Addr</th>
<th>Block Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>

Cache

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

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:

- **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

\[
m \text{-bit address: } \begin{array}{c}
\text{Tag (} t \text{)} \\
\text{Index (} s \text{)} \\
\text{Offset (} k \text{)}
\end{array}
\]

- **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: \((\text{block address}) \mod \text{(\# of blocks in cache)}\)

- 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?
**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*

<table>
<thead>
<tr>
<th>1-way:</th>
<th>2-way:</th>
<th>4-way:</th>
<th>8-way:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 sets, 1 block each</td>
<td>4 sets, 2 blocks each</td>
<td>2 sets, 4 blocks each</td>
<td>1 set, 8 blocks</td>
</tr>
</tbody>
</table>

- Direct mapped

- Fully associative
### 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

---

<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 ($s$)</td>
<td>Offset ($k$)</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: \( \text{0b 0001 1000 0011 0011} \)

  \[
  t = m - s - k \quad \text{Index (s)} \quad \text{Offset (k)}
  \]

  \[
  m = 16 \text{-bit address:} \quad \text{Tag} (t) \quad \text{Index} (s) \quad \text{Offset} (k)
  \]

  \[
  s = \log_2 (C/K/E) \quad s = \log_2 (8) = 3 \quad \text{bits} \quad \text{Direct-mapped (E=1)}
  \]

  \[
  s = \log_2 (8/2) = 2 \quad \text{bits} \quad \text{2-way set associative (E=2)}
  \]

  \[
  s = \log_2 (8/4) = 1 \quad \text{bit} \quad \text{4-way set associative (E=4)}
  \]
Example Placement

Where would data from address \(0x1833\) be placed?

- Binary: \(0b\ 0001\ 1000\ 0011\ 0011\)

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

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

\(m\)-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”)

<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>
</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>
</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>
</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>
</tbody>
</table>

2-way set associative

4-way set associative
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)

  \[ \rightarrow 12's \text{ block evicted } 10's \text{ block} \]
  \[ \rightarrow 10 \text{ and } 12 \text{ are not in the same block} \]
  \[ \rightarrow \text{pulls block containing } 10 \text{ into } $ \]

- **Associativity?**
  \[ N=1 \text{ (direct-mapped)} \]
  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 \( K=1 \) B.
General Cache Organization \((S, E, K)\)

- \(E = \text{blocks/lines per set}\)
- \(S = \# \text{sets} = 2^s\)
- \(K = \text{bytes per block}\)
- \(C = K \times E \times S\) 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 \text{ in book}) )</td>
<td>( M = 2^m \text{ ↔ } m = \log_2 M )</td>
</tr>
<tr>
<td>Cache size</td>
<td>( C )</td>
<td>( S = 2^s \text{ ↔ } s = \log_2 S )</td>
</tr>
<tr>
<td>Associativity</td>
<td>( E )</td>
<td>( K = 2^k \text{ ↔ } 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 \text{ in book}) )</td>
<td></td>
</tr>
</tbody>
</table>