18-447 Lecture 15: Cache Design (in Isolation)

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today
  – recover from Spring Break
  – understand “aBC” of caches
  – understand “3 C’s” of caches

• Notices
  – Lab 3, due next week
  – HW4 out on Wed
  – Midterm 1 regrade submit to Course Hub by Friday

• Readings
  – P&H Ch 5
The Basic Problem

• Potentially $M=2^m$ bytes of memory, how to keep “copies” of most frequently used locations in $C$ bytes of fast storage where $C << M$

• Basic issues (intertwined)
  (1) when to cache a “copy” of a memory location
  (2) where in fast storage to keep the “copy”
  (3) how to find the “copy” later on (*LW and SW only give indices into $M$*)
Direct-Mapped Cache (v1)

**lg₂M-bit address**

| tag | idx | g |

**lg₂(C/G) bits**

| t bits |

**t bits**

What about writes?

**let t= lg₂M−lg₂C**

**Tag Bank**

C/G lines by t bits

valid

**Data Bank**

C/G lines by G bytes

**hit?**

**data**

**G bytes**
Storage Overhead and Block Size

- For each cache block of $G$ bytes, also storing “$t+1$” bits of tag (where $t = \log_2 M - \log_2 C$
  - if $M = 2^{32}$, $G = 4$, $C = 16K = 2^{14}$
  $\Rightarrow t = 18$ bits for each 4-byte block
  60% overhead; 16KB cache actually 25.5KB SRAM

- Solution: “amortize” tag over larger $B$-byte block
  - manage $B/G$ consecutive words as indivisible unit
  - if $M = 2^{32}$, $B = 16$, $G = 4$, $C = 16K$
  $\Rightarrow t = 18$ bits for each 16-byte block
  15% overhead; 16KB cache actually 18.4KB SRAM
  - spatial locality also says this is a good $(Q1: \text{when})$

- Larger caches wants even bigger blocks
Direct-Mapped Cache (final)

let \( t = \lg_2 M - \lg_2 C \)
Is this okay?

let \( t = \lg_2(M) - \lg_2(C) \)
Is this okay?

let \( t = \log_2 M - \log_2 C \)

\[ \log_2(C/B)+k \text{ bits} \]

\[ \log_2(B/G)-k \text{ bits} \]

\[ 2^k C/B \text{-by-} B/2^k \text{ bytes} \]

\[ G \text{ bytes} \]

\[ \text{data} \]
Direct-Mapped Cache in Essence

- **C** bytes of storage managed as **C/B** cache blocks
- A given block address **directly maps** to exactly one choice of cache block (by block index field)
- Block addresses with same block index field map to same cache block
  - of \(2^t\) such addresses, hold only one at a time
  - even if **C** > working set size, conflict is possible
    (“working set” is not one continuous region)
  - probability 2 random addresses conflict is \(1/(C/B)\); likelihood for conflict decreases with increasing number of blocks

\[
\text{hit rate} = \frac{\text{working set size (W)}}{C}
\]
Now for the “general” case
Set Associative Cache

| tag | idx | bo | g |

```
Tag
C/a/B by t bits
valid
```

```
Data
C/a/B by B bytes
```

```
Tag
C/a/B by t bits
valid
```

```
Data
C/a/B by B bytes
```

\[
t = \lg_2 M - \lg_2 (C/a)
\]

direct-mapped

some kind of “mux”
a-way Set-Associative Cache

- C bytes of storage divided into a direct-mapped banks (aka “ways”)
  - each “way” has \((\frac{C}{a})/B\) cache blocks
  - a given block address maps to exactly one choice per “way”; a choices constitute the “set”

  direct-mapped is special case \(a=1\)
  - overhead: a comparators and a-to-1 multiplexer

- Block addresses with same index map to same set
  - \(2^t\) such addresses; hold a different ones at a time
  - if \(C >\) working set size

  higher-degree of associativity \(\Rightarrow\) fewer conflicts

What if \(C <\) working set size?
Replacement Policy to Choose from a

- New block displaces an existing block from “set”
  - pick the one that is least recently used (LRU)
    - exactly LRU expensive for \( a > 2 \)
  - pick any one except the most recently used
  - pick the most recently used one
  - pick one based on some part of the address bits
  - pick the one used again furthest in the future
  - pick a (pseudo) random one

- No real best choice; second-order impact only
  - if actively using less than \( a \) blocks in a set, any sensible replacement policy will quickly converge
  - if actively using more than \( a \) blocks in a set, no replacement policy can help you
“Pseudo”-Associative Cache

• Associativity is a placement policy
  – it says a block address could be placed in one of a different blocks
  – it doesn’t say “ways” are parallel look-up banks

• Pseudo a-way associativity:
  – given a direct-mapped array with $C/B$ blocks
  – logically partition into $C/B/a$ sets
  – given an address $A$, index into set and sequentially search its ways:

• Optimization: record the most recently used way (MRU) to check first
Skewed Associative Cache

t = \log_2 M - \log_2(C/a)

data

C/a byte direct-mapped

different hash for each way

C/a byte

C/a/B by t bits

C/a/B by B bytes

C/a/B by t bits

C/a/B by B bytes

hit?

data

18-447-S19-L15-S15, James C. Hoe, CMU/ECE/CALCM, ©2019
Fully Associative Cache: $a \equiv C/B$

Let $t = \lg_2 M - \lg_2 B$
Fully Associative Cache: $a=C/B$

- A “content-addressable” memory
  - no index bits used in lookup
  - present tag to find a block with matching tag, or else miss
- Any block address can go into any of $C/B$ cache blocks
  - if $C >$ working set size, no conflicts
- Requires 1 comparator per cache block, a huge multiplexer, and many long wires
  - expensive/difficult for more than a few tens of blocks at L1 speed
  - few reasons for very large fully assoc. caches

hit rate ~5
3C’s of Cache Misses
Compulsory Miss

- First reference to a block address always misses (if no prefetching)
- Dominates when locality is poor
  - for example, in a “streaming” data access pattern where many addresses are visited, but each is used only once
- Main design factor: $B$ and “prefetching”
Capacity Miss

• Cache is too small to hold everything needed
• Defined as the misses that would occur in a fully-associative cache of the same capacity using optimum (Belady) replacement
• Dominates when $C < W$
  – for example, the L1 cache usually not big enough due to cycle-time tradeoff
• Main design factor: $C$
Conflict Miss

- Miss to a previously visited block address displaced due to conflict under direct-mapped or set-associative allocation.
- Defined as “a miss that is neither compulsory nor capacity”.
- Dominates when $C \approx W$ or when $C/B$ is small.
- Main design factor: $a$.

\[
\text{hit rate} \approx 5
\]
### 3’C worksheet: $a=1$, $B=1$, $C=2$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>compulsory</td>
<td>$[-,-] \rightarrow [0,-]$</td>
<td>${ } \rightarrow {0}$</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3′C worksheet: \(a=1, \ B=1, \ C=2\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>compulsory</td>
<td>[-,-] → [0,-]</td>
<td>({} \rightarrow {0})</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td>compulsory</td>
<td>[0,-] → [2,-]</td>
<td>({0} \rightarrow {0,2})</td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td>conflict</td>
<td>[2,-] → [0,-]</td>
<td>({0,2}_{\text{hit}})</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td>conflict</td>
<td>[0,-] → [2,-]</td>
<td>({0,2}_{\text{hit}})</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>compulsory</td>
<td>[2,-] → [2,1]</td>
<td>({0,2} \rightarrow {0,1})</td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td>conflict</td>
<td>[2,1] → [0,1]</td>
<td>({0,1}_{\text{hit}})</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td>capacity</td>
<td>[0,1] → [2,1]</td>
<td>({0,1} \rightarrow {0,2})</td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
<td>conflict</td>
<td>[2,1] → [0,1]</td>
<td>({0,2}_{\text{hit}})</td>
</tr>
</tbody>
</table>
Recap: Basic Cache Parameters

- **M = 2^m**: size of address space in bytes
  - Sample values: $2^{32}$, $2^{64}$

- **G = 2^g**: cache access granularity in bytes
  - Sample values: 4, 8

- **C**: “capacity” of cache in bytes
  - Sample values: 16 KByte (L1), 1 MByte (L2)

- **B = 2^b**: “block size” in bytes
  - Sample values: 16 (L1), >64 (L2)

- **a**: “associativity” of the cache
  - Sample values: 1, 2, 4, 5(?),... “C/B”

C/a should be a 2-power
Recap: Address Fields

\[ \log_2 M \text{-bit address} \]

- **tag**: \( \log_2 M \) bits
- **index**: \( \log_2 (C/a) \) bits
- **B.O.**: \( \log_2 (B/G) \) bits
- **block offset**: \( \log_2 G \) bits
- **byte offset**: \( \log_2 G \) bits

[Diagram showing the structure of an address field with the components labeled as described above.]
M=2^{32}, \ a=2, \ C=1K, \ B=4, \ G=2
**M=2^{32}, a=2, C=1\text{K}, B=4, G=2: basic solution**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>idx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.o.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \text{idx}=7
- \text{tag0}: 128-l \times 23-b
- \text{v0}: 1-b
- \text{tag1}: 128-l \times 23-b
- \text{v1}: 1-b
- \text{tag}=23
- \text{hit0}=\text{hit1}

\text{data 0}: 128\text{-lines} \times 4\text{-bytes}

\text{data 1}: 128\text{-lines} \times 4\text{-bytes}

\text{hit0}, \text{hit1} \rightarrow \text{2-1-mux} \rightarrow \text{DATA}

18-447-S19-L15-S27, James C. Hoe, CMU/ECE/CALCM, ©2019
Can you play the same trick on the tag SRAMs?

Same cache parameters but tune for “narrower” data SRAMs

tag


idx

b.o.

this part is unchanged

Can you play the same trick on the tag SRAMs?
Can you play the same trick on the tag SRAMs?
Same cache parameters but each block frame is interleaved over 2 SRAM banks

<table>
<thead>
<tr>
<th>tag</th>
<th>idx</th>
<th>b.o.</th>
</tr>
</thead>
</table>

```
idx
↓ 7
```

```
tag0  v0
128-l x 23-b
```

```
tag1  v1
128-l x 23-b
```

```
tag
```

```
23
```

```
h0  h1
```

```
2-1-mux
```

```
2-1-muxd
```

```
data 0
128-lines x 4-bytes
```

```
data 1
128-lines x 4-bytes
```

```
h0•bo  h1•bo
```

```
h1•bo  h0•bo
```

```
HIT
```

```
16
```

this part is unchanged