Caches

Han Wang
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H 5.1, 5.2 (except writes)
Announcements

This week:
• PA2 Work-in-progress submission

Next six weeks:
• Two labs and two projects
• Prelim2 will be Thursday, March 29th
Goals for Today: caches

Caches vs memory vs tertiary storage

- Tradeoffs: big & slow vs small & fast
  - Best of both worlds
- working set: 90/10 rule
- How to predict future: temporal & spacial locality

Cache organization, parameters and tradeoffs

- associativity, line size, hit cost, miss penalty, hit rate
  - Fully Associative $\rightarrow$ higher hit cost, higher hit rate
  - Larger block size $\rightarrow$ lower hit cost, higher miss penalty
Agenda

- Memory Hierarchy Overview
- The Principle of Locality
- Direct-Mapped Cache
- Fully Associative Cache
## Performance

CPU clock rates $\sim 0.2\text{ns} – 2\text{ns}$ (5GHz-500MHz)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Capacity</th>
<th>$$/\text{GB}$$</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>1 TB</td>
<td>$.17</td>
<td>100s of seconds</td>
</tr>
<tr>
<td>Disk</td>
<td>2 TB</td>
<td>$.03</td>
<td>Millions of cycles (ms)</td>
</tr>
<tr>
<td>SSD (Flash)</td>
<td>128 GB</td>
<td>$2</td>
<td>Thousands of cycles (us)</td>
</tr>
<tr>
<td>DRAM</td>
<td>8 GB</td>
<td>$10</td>
<td>50-300 cycles (10s of ns)</td>
</tr>
<tr>
<td>SRAM off-chip</td>
<td>8 MB</td>
<td>$4000</td>
<td>5-15 cycles (few ns)</td>
</tr>
<tr>
<td>SRAM on-chip</td>
<td>256 KB</td>
<td>???</td>
<td>1-3 cycles (ns)</td>
</tr>
</tbody>
</table>

Others: eDRAM aka 1-T SRAM, FeRAM, CD, DVD, ...

Q: Can we create illusion of cheap + large + fast?
Memory Pyramid

RegFile  
100s bytes

< 1 cycle access

L1 Cache  
(several KB)

1-3 cycle access

L2 Cache  
(½-32MB)

5-15 cycle access

Memory  
(128MB – few GB)

50-300 cycle access

Disk  
(Many GB – few TB)

1000000+ cycle access

These are rough numbers: mileage may vary for latest/greatest Caches usually made of SRAM (or eDRAM)
A Typical System
Memory Hierarchy

Memory closer to processor
- small & fast
- stores active data

Memory farther from processor
- big & slow
- stores inactive data
Active vs Inactive Data

Assumption: Most data is not active.

Q: How to decide what is active?

A: Programmer decides

A: Compiler decides

A: OS decides at run-time

A: Hardware decides at run-time
Insight of Caches

Q: What is “active” data?

If Mem[x] is was accessed *recently*...

... then Mem[x] is likely to be accessed *soon*

• Exploit temporal locality:

... then Mem[x ± ε] is likely to be accessed *soon*

• Exploit spatial locality:
Locality Analogy

- Writing a report on a specific topic.
- While at library, check out books and keep them on desk.
- If need more, check them out and bring to desk.
- But don’t return earlier books since might need them
- Limited space on desk; Which books to keep?
- You hope this collection of ~20 books on desk enough to write report, despite 20 being only 0.00002% of books in Cornell libraries
Two types of Locality

Temporal Locality (locality in time)

• If a memory location is referenced then it will tend to be referenced again soon

⇒ Keep most recently accessed data items closer to the processor

Spatial Locality (locality in space)

• If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

⇒ Move blocks consisting of contiguous words closer to the processor
Locality

```c
int n = 4;
int k[] = { 3, 14, 0, 10 };

int fib(int i) {
    if (i <= 2) return i;
    else return fib(i-1)+fib(i-2);
}

int main(int ac, char **av) {
    for (int i = 0; i < n; i++) {
        printf(fib(k[i]));
        printf("\n");
    }
}
```
Memory Hierarchy

Memory closer to processor is fast but small

- usually stores subset of memory farther away
  - “strictly inclusive”

- alternatives:
  - strictly exclusive
  - mostly inclusive

- Transfer whole blocks (cache lines):
  - 4kb: disk ↔ ram
  - 256b: ram ↔ L2
  - 64b: L2 ↔ L1
Cache Lookups (Read)

Processor tries to access Mem[x]

Check: is block containing Mem[x] in the cache?

• Yes: cache hit
  – return requested data from cache line

• No: cache miss
  – read block from memory (or lower level cache)
  – (evict an existing cache line to make room)
  – place new block in cache
  – return requested data

→ and stall the pipeline while all of this happens
Cache has to be fast and dense

- Gain speed by performing lookups in parallel
  - but requires die real estate for lookup logic
- Reduce lookup logic by limiting where in the cache a block might be placed
  - but might reduce cache effectiveness
Three common designs

A given data block can be placed...

- ... in any cache line → Fully Associative
- ... in exactly one cache line → Direct Mapped
- ... in a small set of cache lines → Set Associative
Three common designs

A given data block can be placed...

- ... in any cache line → Fully Associative
- ... in exactly one cache line → Direct Mapped
- ... in a small set of cache lines → Set Associative
Three common designs

A given data block can be placed...

- ... in any cache line $\rightarrow$ Fully Associative
- ... in exactly one cache line $\rightarrow$ Direct Mapped
- ... in a small set of cache lines $\rightarrow$ Set Associative

![Diagram of memory and cache lines with set-associative mapping]

Memory

2-way set-associative $\rightarrow$

way0

way1
TIO: Mapping the Memory Address

- Lowest bits of address (Offset) determine which byte within a block it refers to.
- Full address format:
  - n-bit Offset means a block is how many bytes?
  - n-bit Index means cache has how many blocks?
Direct Mapped Cache

- Each block number mapped to a single cache line index
- Simplest hardware

<table>
<thead>
<tr>
<th>line 0</th>
<th>line 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Direct Mapped Cache

- Each block number mapped to a single cache line index
- Simplest hardware

<table>
<thead>
<tr>
<th>line 0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
<td></td>
</tr>
<tr>
<td>line 2</td>
<td></td>
</tr>
<tr>
<td>line 3</td>
<td></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
</tr>
<tr>
<td>0x000004</td>
</tr>
<tr>
<td>0x000008</td>
</tr>
<tr>
<td>0x00000c</td>
</tr>
<tr>
<td>0x000010</td>
</tr>
<tr>
<td>0x000014</td>
</tr>
<tr>
<td>0x000018</td>
</tr>
<tr>
<td>0x00001c</td>
</tr>
<tr>
<td>0x000020</td>
</tr>
<tr>
<td>0x000024</td>
</tr>
<tr>
<td>0x000028</td>
</tr>
<tr>
<td>0x00002c</td>
</tr>
<tr>
<td>0x000030</td>
</tr>
<tr>
<td>0x000034</td>
</tr>
<tr>
<td>0x000038</td>
</tr>
<tr>
<td>0x00003c</td>
</tr>
<tr>
<td>0x000040</td>
</tr>
<tr>
<td>0x000044</td>
</tr>
<tr>
<td>0x000048</td>
</tr>
</tbody>
</table>
Tags and Offsets

Assume sixteen 64-byte cache lines

0x7FFF3D4D

= 0111 1111 1111 1111 0011 1101 0100 1101

Need meta-data for each cache line:

- valid bit: is the cache line non-empty?
- tag: which block is stored in this line (if valid)

Q: how to check if X is in the cache?
Q: how to clear a cache line?
A Simple Direct Mapped Cache

Using **byte addresses** in this example! Addr Bus = 5 bits

Processor

<table>
<thead>
<tr>
<th>$1</th>
<th>$2</th>
<th>$3</th>
<th>$4</th>
</tr>
</thead>
<tbody>
<tr>
<td>lb</td>
<td>M[0]</td>
<td>lb</td>
<td>M[6]</td>
</tr>
<tr>
<td>lb</td>
<td>M[10]</td>
<td>lb</td>
<td>M[12]</td>
</tr>
<tr>
<td>$1</td>
<td></td>
<td>$2</td>
<td></td>
</tr>
<tr>
<td>$3</td>
<td></td>
<td>$4</td>
<td></td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
<td>107</td>
<td>109</td>
<td>113</td>
<td>127</td>
<td>131</td>
<td>137</td>
<td>139</td>
<td>149</td>
<td>151</td>
<td>157</td>
<td>163</td>
<td>167</td>
<td>173</td>
<td>179</td>
<td>181</td>
</tr>
</tbody>
</table>

Direct Mapped Cache

V | | | | |
tag | | | |
data | | | |

A =

Hits: 2  Misses: 6
Direct Mapped Cache (Reading)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
</table>

Diagram:

```
V  Tag  Block
```

hit : ?
Direct Mapped Cache Size

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
</table>

$n$ bit index, $m$ bit offset

Q: How big is cache (data only)?

Q: How much SRAM needed (data + overhead)?
Cache Performance

Cache Performance (very simplified):

L1 (SRAM): 512 x 64 byte cache lines, direct mapped
Data cost: 3 cycle per word access
Lookup cost: 2 cycle
Mem (DRAM): 4GB
Data cost: 50 cycle per word, plus 3 cycle per consecutive word

\[ T = T_{hit} \times \% + T_{miss} \]

\[ T_{hit} = 5 \]

\[ T_{miss} = 5 + 50 + 15 \times 3 \]

Performance depends on:
Access time for hit, miss penalty, hit rate
Misses

Cache misses: classification

The line is being referenced for the first time
  • Cold (aka Compulsory) Miss

The line was in the cache, but has been evicted
Avoiding Misses

Q: How to avoid...

Cold Misses
- Unavoidable? The data was never in the cache...
- Prefetching!

Other Misses
- Buy more SRAM
- Use a more flexible cache design
Bigger cache doesn’t always help...
Mem access trace: 0, 16, 1, 17, 2, 18, 3, 19, 4, ...
Hit rate with four direct-mapped 2-byte cache lines?

With eight 2-byte cache lines?

With four 4-byte cache lines?
Misses

Cache misses: classification

The line is being referenced for the first time
  • Cold (aka Compulsory) Miss

The line was in the cache, but has been evicted...
  ... because some other access with the same index
    • Conflict Miss

... because the cache is too small
  • i.e. the *working set* of program is larger than the cache
    • Capacity Miss
Avoiding Misses

Q: How to avoid...

Cold Misses
  • Unavoidable? The data was never in the cache...
  • Prefetching!

Capacity Misses
  • Buy more SRAM

Conflict Misses
  • Use a more flexible cache design
Three common designs

A given data block can be placed...

- ... in any cache line → Fully Associative
- ... in exactly one cache line → Direct Mapped
- ... in a small set of cache lines → Set Associative
A Simple Fully Associative Cache

Using **byte addresses** in this example! Addr Bus = 5 bits

### Processor

- \( \text{lb } \$1 \leftarrow M[1] \)
- \( \text{lb } \$2 \leftarrow M[13] \)
- \( \text{lb } \$3 \leftarrow M[0] \)
- \( \text{lb } \$3 \leftarrow M[6] \)
- \( \text{lb } \$2 \leftarrow M[5] \)
- \( \text{lb } \$2 \leftarrow M[6] \)
- \( \text{lb } \$2 \leftarrow M[10] \)
- \( \text{lb } \$2 \leftarrow M[12] \)

### Memory

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>103</td>
<td>107</td>
<td>109</td>
<td>113</td>
<td>127</td>
<td>131</td>
<td>137</td>
<td>139</td>
<td>149</td>
<td>151</td>
<td>157</td>
<td>163</td>
<td>167</td>
<td>173</td>
<td>179</td>
<td>181</td>
</tr>
</tbody>
</table>

### Fully Associative Cache

\( A = \)

<table>
<thead>
<tr>
<th>V</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hits:**

**Misses:**
Fully Associative Cache (Reading)

- Tag
- Offset
- V Tag
- Block
- hit?
- line select
- word select
- 64 bytes
- 32 bits
- data
Fully Associative Cache Size

<table>
<thead>
<tr>
<th>Tag</th>
<th>Offset</th>
</tr>
</thead>
</table>

$m$ bit offset, $2^n$ cache lines

Q: How big is cache (data only)?
Q: How much SRAM needed (data + overhead)?
Fully-associative reduces conflict misses...
... assuming good eviction strategy
Mem access trace: 0, 16, 1, 17, 2, 18, 3, 19, 4, 20, ...
Hit rate with four fully-associative 2-byte cache lines?
... but large block size can still reduce hit rate
vector add trace: 0, 100, 200, 1, 101, 201, 2, 202, ...
Hit rate with four fully-associative 2-byte cache lines?

With two fully-associative 4-byte cache lines?
Misses

Cache misses: classification

Cold (aka Compulsory)
  • The line is being referenced for the first time

Capacity
  • The line was evicted because the cache was too small
  • i.e. the *working set* of program is larger than the cache

Conflict
  • The line was evicted because of another access whose index conflicted
Summary

Caching assumptions

• small working set: 90/10 rule
• can predict future: spatial & temporal locality

Benefits

• big & fast memory built from (big & slow) + (small & fast)

Tradeoffs:

  associativity, line size, hit cost, miss penalty, hit rate

• Fully Associative → higher hit cost, higher hit rate
• Larger block size → lower hit cost, higher miss penalty

Next up: other designs; writing to caches