Cache Memory Mapping Techniques

Continue to read pp. 289-305
Cache Memory Mapping

• Again cache memory is a small and fast memory between CPU and main memory
• A block of words have to be brought in and out of the cache memory continuously
• Performance of the cache memory mapping function is key to the speed
• There are a number of mapping techniques
  – Direct mapping
  – Associative mapping
  – Set associative - mapping
Direct Mapping Technique – No. 1

- Simplest way of mapping
- Main memory is divided in blocks
- Block j of the main memory is mapped onto block \( j \mod 128 \) of the cache – consider a cache of 128 blocks of 16 words each

<table>
<thead>
<tr>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
</tr>
<tr>
<td>tag</td>
</tr>
<tr>
<td>tag</td>
</tr>
</tbody>
</table>

- Consider a memory of 64K words divided into 4096 blocks

<table>
<thead>
<tr>
<th>Tag</th>
<th>Block</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Main memory address

Where blocks 0, 128, 256, … 3968 should be mapped to?
Where blocks 126, 254, 382, … 4094 should be mapped to?
Direct Mapping Technique (Continued)

• Mapping process
  – Use tag to see if a desired word is in cache
  – If there is no match, the block containing the required word must first be read from the memory
  – For example: MOVE $A815, DO

\[
10101 \hspace{1cm} 0000001 \hspace{1cm} 0101
\]

  Tag  \hspace{1cm}  Block \# \hspace{1cm}  Word

a. Check if cache has tag 10101 for block 1
   match -> hit; different -> miss, load the corresponding block
b. Access word 5 of the block
Direct Mapping Technique (Continued)

• Advantage
  – simplest replacement algorithm

• Disadvantage
  – not flexible
  – there is contention problem even when cache is not full
    • For example, block 0 and block 128 both take only block 0 of cache:
      – 0 modulo 128 = 0
      – 128 modulo 128 = 0
      – If both blocks 0 and 128 of the main memory are used a lot, it will be very slow
Associative Mapping Technique – No. 2

• Any block can go anywhere in cache
• 4095 blocks -> 4095 tag = $2^{12}$ -> 12 bit tag

<table>
<thead>
<tr>
<th>Cache</th>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Block 0</td>
</tr>
<tr>
<td>tag</td>
<td>Block 1</td>
</tr>
<tr>
<td>tag</td>
<td>Block 127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>

Main memory address
Associative Mapping Technique (continued)

• Advantage
  – Any empty block in cache can be used, flexible
  – Must check all tags to check for a hit, expensive
    (parallel algorithm has been developed to speed up the process)

• What is the next technique?
  – Something between direct mapping and associative mapping
Set Associative Mapping Technique – No. 3

- Comprise between direct mapping and associative mapping
- Block in main memory maps to a set of blocks in cache – direct mapping
- Can map to any block within the set
- E.g. use 6 bits for tag = \(2^6 = 64\) tags
- 6 bits for set = \(2^6 = 64\) sets
Set Associative Mapping Technique (continued)

- Memory Address

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Set 1</th>
<th>Set 63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Block 0</td>
<td>Tag</td>
</tr>
<tr>
<td>Block 1</td>
<td>Tag</td>
<td>Block 126</td>
</tr>
<tr>
<td>Block 2</td>
<td>Tag</td>
<td>Block 127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Set</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

- The blocks in cache are divided into 64 sets and there are two blocks in each set
- How the blocks in the main memory be mapped into cache?
- Main memory blocks 0, 64, 128, 4032 maps to set 0 and can occupy either of the two positions
Set Associative Mapping Technique (continued)

- A set could have **one** block -> **direct mapping**; 128 blocks -> **associative mapping**
- **k** blocks per set is referred to as **k-way set-associative mapping**

```
Main memory

Tag 0
---
Block 0
Block 63
Block 64
Tag 1
---
Block 127
Tag 63
---
Block 4032
Block 4095
```

Set 0
Cache Memory Details

• Block size
  – Depends on how memory is addressed (byte, word, or long word) and accessed (word at a time)
  – 8-16 quite reasonable
    • 68040 – 16 bytes per block
    • Pentium IV – 64 bytes per block
  – Always work with 1 block at a time
  – How many blocks in cache?
    • No of words in cache divided by number of words per block – e.g. 2 k words, 16-word block: \( \frac{2^{11}}{2^4} = 2^7 = 128 \) blocks
Cache Memory Details (continued)

• Replacement Algorithms
  – Replace the one that has gone the longest time without being referenced – Least Recently Used (LRU) – block

• How to know which block of main memory is currently in cache?
  – Look at the tag on data in the block
  – How long is the tag (how many blocks use same block of cache)?

• Study a few examples
Examples

- Small Instruction Cache (read 8.6.3)
  - Cache has 8 blocks, 1 word each
  - Main memory has 256 blocks (words) – 8 bit address
  - Execute the following program
  - Use direct mapping first

```
C0  C1  D0  D1  D2  E0
```

Loop 10 times

Loop 5 times

```
Block 0
Block 1
Block 7
```

```
Tag Block
5  3
```

Main memory address
Direct Mapping Performance

• How many executions? - \((2 \times 10 + 4) \times 5 = 120\)

<table>
<thead>
<tr>
<th>Cache Block</th>
<th>After C1</th>
<th>After Inner Loop</th>
<th>After E0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C0</td>
<td>D0</td>
<td>E0</td>
</tr>
<tr>
<td>1</td>
<td>C1</td>
<td>D1</td>
<td>D1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>D2</td>
</tr>
</tbody>
</table>

Misses: \(2 \times 5\) \(2 \times 5\) \(2 + 1 \times 4 = 26\)

Hits: \(18 \times 5\) \(1 \times 4 = 94\)

• Hit rate = hits/total = 94/120 = 78.3%
## Associative Mapping Performance

<table>
<thead>
<tr>
<th>Cache Block</th>
<th>After C1</th>
<th>After Inner Loop</th>
<th>After E0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C0</td>
<td>C0</td>
<td>C0</td>
</tr>
<tr>
<td>1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>D0</td>
<td>D0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>D1</td>
<td>D1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>D2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>E0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Misses
- 2
- 2
- 2

\[ \text{Misses} = 2 + 2 + 2 = 6 \]

### Hits
- next 4 times all hits = 114

- Hit rate = hits/total = 114/120 = 95%
## Set Associative Performance

2-way $\rightarrow$ 4 sets

<table>
<thead>
<tr>
<th>Cache Block</th>
<th>After C1</th>
<th>After Inner Loop</th>
<th>After E0</th>
<th>Second time After C1</th>
<th>After Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td>0</td>
<td>C0</td>
<td>C0</td>
<td>E0</td>
<td>E0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>D0</td>
<td>D0</td>
<td>C0</td>
</tr>
<tr>
<td>Set 1</td>
<td>0</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
</tr>
<tr>
<td>Set 2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Misses \[ 2 + 1 \times 4 \] \[ 2 + 1 \times 4 \] \[ 2 + 1 \times 4 \] = 18

Hits The rest is all hits = 102

- Hit rate = hits/total = 102/120 = 85%