Optimizing Cache Performance
Today’s Lecture

- Use larger cache blocks to take advantage of spatial locality
- Use set associativity to resolve “hashing collisions”
For a byte-addressable machine with 16-bit addresses

Which picture best represents a cache that is direct-mapped, each block holds one byte, and has eight cache blocks
For a byte-addressable machine with 16-bit addresses

A cache has the following characteristics:

- It is direct-mapped (as discussed last time)
- Each block holds one byte
- The cache index is the four least significant bits

Two questions:

- How many blocks does the cache hold?
  a) 1  
  b) 2  
  c) 4  
  d) 8  
  e) 16

- How many bits are stored at each cache block (e.g., for the data array, tags, etc.)?
  a) 8  
  b) 9  
  c) 12  
  d) 20  
  e) 21
1-byte cache blocks do not take advantage of spatial locality.

Create larger cache blocks.
Creating larger cache blocks moves adjacent blocks as a unit

16-byte main memory

Two bytes (12 and 13) move as a unit

8-byte cache
Each cache block is an array indexed by a block offset

Create an “array of arrays”

8-byte cache

Block offset
0
1

Block Index
0
1
2
3
The least-significant bit(s) is used to index the block offset "array"
Data gets moved in cache blocks and not bytes

Consider the following set of memory loads:

<table>
<thead>
<tr>
<th>Load</th>
<th>Block Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>1</td>
</tr>
</tbody>
</table>

What is stored in block index 01, offset 1?

- a) 201
- b) 210
- c) 211
- d) 215
- e) invalid
To increase cache block size, subdivide an m-bit address into tag, index, and block offset

- Suppose we have a cache with $2^k$ blocks, each containing $2^n$ bytes.
  - Lowest $n$ bits are the **block offset** that decides which of the $2^n$ bytes in the cache block will store the data.
  - Next $k$ bits of the address select one of the $2^k$ cache blocks.

- Example: $2^2$-block cache with $2^1$ bytes per block. Memory address 13 (1101) would be stored in offset 1 of cache block 2.

```
<table>
<thead>
<tr>
<th>1-bit tag</th>
<th>2 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
```
Implement block offset with a multiplexer
For the addresses below, what byte is read from the cache (or is there a miss)?

- 1010: Miss, tag mismatch
- 1110: Miss, invalid
- 0001: Hit, 0xDE
- 1101: Hit, 0xEF
- 1101: Hit, 0xFE
For a byte-addressable machine with 16-bit addresses

A cache has the following characteristics:

- It is direct-mapped
- Each block holds four bytes
- The cache has 32 cache blocks

How many bits are used for the tag?

a) 2  
b) 4  
c) 5  
d) 7  
e) 9
Direct-mapped caches fall short when addresses collide

- Example: what happens if a program uses addresses 2, 6, 2, 6, 2, ...?

THRASHING!!
A **fully-associative cache** permits data to be stored in *any* cache block

- When data is fetched from memory, it can be placed in *any* unused block of the cache.
- Maximize temporal locality by keeping the most recently used data in the cache, replace the **least-recently used (LRU)** when cache is full.
A fully associative cache can map addresses anywhere, eliminating thrashing

- Example: what happens if a program uses addresses 2, 6, 2, 6, 2, ...?
  M, M, H, H, H
A fully-associative cache is expensive because we need to store the entire tag!

- Data could be anywhere in the cache, so we must check the tag of every cache block. That’s a lot of comparators!