The Memory Hierarchy (3)

(Slides adapted from CSAPP)
Summary of previous class

- **Memory hierarchy**
  - Temporal and spatial locality
  - Cache organization and addressing

- **This class: More on memory hierarchy**
  - Cache addressing
  - Cache friendly programming
Agenda

- Caches organization
  - Direct Mapped Cache
  - 2-set associative
  - Fully associative

- Cache performance impact
  - Memory mountain

- Cache friendly programming
  - Rearranging the loops to improve locality
  - Using blocking to improve temporal locality
Cache memory organization and operation
Cache Read (Review)

- Locate set
- Check if any line in set has matching tag
- Yes + line valid: hit
- Locate data starting at offset

\[ E = 2^e \] lines per set

\[ S = 2^s \] sets

\[ B = 2^b \] bytes per cache block (the data)

Address of word:
- \( t \) bits
- \( s \) bits
- \( b \) bits

- tag
- set
- index
- block
- offset

data begins at this offset

valid bit

Locate set
Check if any line in set has matching tag
Yes + line valid: hit
Locate data starting at offset
Example: Direct Mapped Cache ($E = 1$)

Direct mapped: One line per set
Assume: cache block size 8 bytes

$S = 2^s$ sets

```
<table>
<thead>
<tr>
<th>v</th>
<th>tag</th>
<th>0 1 2 3 4 5 6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>
```

Address of int:
```
t bits 0...01 100
```

find set
Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

Address of int:

valid? + match: assume yes = hit

block offset

v  tag  0  1  2  3  4  5  6  7

t bits 0...01 100
Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes

Valid? + match: assume yes = hit

Address of int:

\[
\begin{array}{c|c|c}
\text{t bits} & 0\ldots01 & 100 \\
\end{array}
\]

Block offset

int (4 Bytes) is here

No match: old line is evicted and replaced

Why alignment?
- To ensure one memory access to access a data type
In the above example, if the address of int is started from 6, two memory accesses is required to access it
Why alignment?

- To ensure one memory access to access a data type
- In previous example, if the address of int started from 6, two memory access was required to access it
Direct-Mapped Cache Simulation

M=16 byte addresses, B=2 bytes/block, S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

<table>
<thead>
<tr>
<th>v</th>
<th>Tag</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0000₂],</td>
<td>miss</td>
</tr>
<tr>
<td>1</td>
<td>[0001₂],</td>
<td>hit</td>
</tr>
<tr>
<td>7</td>
<td>[0111₂],</td>
<td>miss</td>
</tr>
<tr>
<td>8</td>
<td>[1000₂],</td>
<td>miss</td>
</tr>
<tr>
<td>0</td>
<td>[0000₂]</td>
<td>miss</td>
</tr>
</tbody>
</table>

Set 0: 1 0  M[0-1]  
Set 1: 
Set 2: 
Set 3: 1 0  M[6-7]
A High Level Example

int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];

    return sum;
}

int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; i < 16; i++)
        for (i = 0; j < 16; j++)
            sum += a[i][j];

    return sum;

}
E-way Set Associative Cache (Here: $E = 2$)

$E = 2$: Two lines per set
Assume: cache block size 8 bytes

Address of short int:

```
t bits 0...01 100
```

find set
E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of short int:

| t bits | 0...01 | 100 |

valid? + match: yes = hit

compare both

block offset
E-way Set Associative Cache (Here: $E = 2$)

$E = 2$: Two lines per set
Assume: cache block size 8 bytes

Address of short int:

<table>
<thead>
<tr>
<th>t bits</th>
<th>0...01</th>
<th>100</th>
</tr>
</thead>
</table>

valid? + match: yes = hit

short int (2 Bytes) is here

No match:
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...
Replacement policy

- **Optimal policy**: Replace the line whose next use will occur farthest in the future
  - It is not feasible to implement, because we have to predict the access pattern of lines

- **LRU**: Replace the line whose recent use is occurred farthest in the past
  - LRU can provide near-optimal performance in theory

- **Random**: Choose a line randomly and replace it
  - Causes random behavior with high (error) variance
2-Way Set Associative Cache Simulation

- \( t=2 \)
- \( s=1 \)
- \( b=1 \)

M=16 byte addresses, B=2 bytes/block,
S=2 sets, E=2 blocks/set

Address trace (reads, one byte per read):

- 0  \([0000_2]\),  miss
- 1  \([0001_2]\),  hit
- 7  \([0111_2]\),  miss
- 8  \([1000_2]\),  miss
- 0  \([0000_2]\)  hit

<table>
<thead>
<tr>
<th>v</th>
<th>Tag</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>M[0-1]</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>M[8-9]</td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>M[6-7]</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

```c
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; i < 16; i++)
        for (i = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables sum, i, j

assume: cold (empty) cache, a[0][0] goes here

32 B = 4 doubles
What about writes?

- Multiple copies of data exist:
  - L1, L2, Main Memory, Disk

- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from memory or not)

- What to do on a write-miss?
  - Write-allocate (load into cache, update line in cache)
    - Good if more writes to the location follow
  - No-write-allocate (writes immediately to memory)

- Typical
  - Write-through + No-write-allocate
  - Write-back + Write-allocate
Quiz 7 review
Which of the following statements are true?

- PROM and EPROM lose information if powered off.
- Between two caches with the same size, the one that has more sets has less conflict miss rate.
- Solid State DISK (SSD) compared to rotating disks is faster and consume the less power.
- In a E set associative cache, the conflict miss rate decreases with increasing the number of lines per each set.
Assume a two level hierarchy memory, such that the access time for the first level is 2 ns, and the access for the second level is 1 ms. What is the average access time for the memory if the hit rate for the first level is 90%.

Avg. access time = 0.9 * 2 + 0.1 * (2 + 1000000)
For the following code describe how temporal locality and spatial locality changes with the small and large values for the step and size.

```
sum = 0;
for (i = 0; i <size; i+=step)
    sum += a[i];
return sum;
```

- Increasing the value for step → decreasing the spatial locality
- Increasing the size → increasing the working set → decreasing temporal locality
How can you improve the locality for the following code?

```
sum = 0;
int A[m][n]
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        sum += a[j][i];
return sum;
```

```
sum = 0;
int A[m][n]
for (j = 0; j < m; j++)
    for (i = 0; i < n; i++)
        sum += a[j][i];
return sum;
```
Calculate the size of the cache memory with the following parameters:

Number of sets: 32; number of lines per each set: 2; and number of bytes per each block of data: 8

Capacity = 32*2*8