Last class

L → Nodes

When to update?

Write-through  Write-back

What to do about misses?

Write-allocate  No-write-allocate

Cache Metrics

- Miss Rate
  \[
  \text{Miss Rate} = \frac{\# \text{ of Misses}}{\# \text{ of References}}
  \]
  ↓ low
Hit rate

\[ \text{Hit rate} = 1 - \text{miss rate} \]

↑ high

Hit time

→ Time to move a word in the cache to the registers (processor).

(includes → time for set line and word within block (selections))

↓ low
as possible.
Miss Penalty.

Additional time because of a miss.

If the miss is on L1, then miss penalty is the time it takes to read the word from L2.

How do different cache parameters affect cache performance?
Impact of Cache Size

- Increase: Better hit rates
- Drawbacks: ↑ hit times

Impact of Block Size

- Increase: Increase the hit rate if programs have good spatial locality
- Drawbacks:
  1. For a given cache capacity, this means fewer lines
  2. Programs with temporal locality do not transfer time well
Impact of Associativity

- Drawbacks:
  1. Increased hit times.
  2. Increased miss penalty (time to choose a victim block).

- Fewer conflict misses due to thrashing.
- Hit rate increased.

Faster memory $\rightarrow$ Smaller associativity

Low hit rate $\rightarrow$ Miss penalty for the next memory is not that high.

Fast hit times $\rightarrow$ High associativity.

Slower memories $\rightarrow$ High associativity.
Impact of write strategies

Write-through vs. Write-back

Simple vs. Not simple
need to store an additional bit

Transfer times

Traffic is going to be high

Cache Friendly Code
Cache Organization
IV

April 1, 2016  .  Ganesh Kumar
Consider this program

```c
int sumarray(int arr[8]) {
    int sum = 0;

    for (int i = 0; i < 8; i++)
        sum += arr[i];

    return sum;
}
```

Good temporal locality?

Good spatial locality?
**Code**

```c
int sumarray(int arr[8]) {
    int sum = 0;

    for (int i = 0; i < 8; i++)
        sum += arr[i];

    return sum;
}
```

Good temporal locality?

YES! Variables `i` and `sum` are accessed repeatedly!

Good spatial locality?

Clearly! Variable `arr` is being accessed sequentially in a stride-1-reference pattern.
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

<table>
<thead>
<tr>
<th>arr[i]</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>
</tr>
</thead>
<tbody>
<tr>
<td>Access order</td>
<td>1[m]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[h] - hit
[m] - miss
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

Now load block with arr[0] onto the cache.
Since block size is 16 Bytes, the first four elements of the array get loaded into the cache.
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

<table>
<thead>
<tr>
<th>arr[i]</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>
</tr>
</thead>
<tbody>
<tr>
<td>Access order</td>
<td>1[m]</td>
<td>2[h]</td>
<td>3[h]</td>
<td>4[h]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[h] - hit
[m] - miss

- Now load block with arr[0] onto the cache.
- Since block size is 16 Bytes, the first four elements of the array get loaded into the cache.
- **Accessing arr[1], arr[2] and arr[3] will now be hits!**
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

<table>
<thead>
<tr>
<th>arr[i]</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>
</tr>
</thead>
<tbody>
<tr>
<td>Access order</td>
<td>1[m]</td>
<td>2[h]</td>
<td>3[h]</td>
<td>4[h]</td>
<td>5[m]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[h] - hit
[m] - miss

- Now the loop will access arr[4].
- Miss!
- Load the block containing arr[4] onto the cache.
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

Now the loop will access arr[4].
- Miss!
- Load the block containing arr[4] onto the cache.
- Since block size is 16 bytes, arr[5], arr[6] and arr[7] will also get loaded.
- And when the loop accesses them, it will be cache hits!
Assume the following,
- arr is block-aligned.
- Words are 4 bytes.
- Block size is 4 words (16 Bytes).
- Cache is initially empty

<table>
<thead>
<tr>
<th>arr[i]</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>
</tr>
</thead>
<tbody>
<tr>
<td>Access order</td>
<td>1[m]</td>
<td>2[h]</td>
<td>3[h]</td>
<td>4[h]</td>
<td>5[m]</td>
<td>6[h]</td>
<td>7[h]</td>
<td>8[h]</td>
</tr>
</tbody>
</table>

[h] - hit
[m] - miss

Miss Rate = # of Misses / # of References
Miss Rate = 2/8 = 0.25
In general, if a cache has a block size of $B$ bytes, then stride-$k$-reference pattern will produce an average of

$$\min \left( 1, \frac{\text{wordsize} \times k}{B} \right)$$

misses for each iteration of the loop

where $k$ is expressed in words.

For our example,

Average misses = $\min \left( 1, \frac{4 \times 1}{16} \right) \Rightarrow \min (1, 0.25) \Rightarrow 0.25$

We can expect an average of 0.25 misses for every iteration.