1. 2.1 Introduction (material covered in previous slide set)

2. 2.2 10 Advanced Optimizations of Cache Performance

3. 2.3 Memory Technology (optional material, not discussed in class)

4. 2.4 Protection: Virtual Memory and Virtual Machines (virtual memory discussed in previous slide set, expanded virtualization covered by outside reading and next slide set).
Categories of Cache Misses

1. Compulsory: First access to a block.
2. Capacity: If the cache cannot hold all the blocks needed during execution, capacity misses will occur.
3. Conflict: When block placement is limited so that conflict occurs between multiple blocks with intermingled accesses.
Categories of Optimization

- Reducing hit time: small and simple caches and way prediction (both techniques also generally decrease power consumption).

- Increasing cache bandwidth: pipelined caches, multi-banked caches, and non-blocking caches (varying impact on power).

- Reducing miss penalty: critical word first and merging write buffers (little impact on power).

- Reducing miss rate: compiler optimizations.

- Reducing miss penalty or miss rate via parallelism: hardware prefetching and compiler prefetching (generally increase power consumption).
1. Small and simple L1 caches to reduce hit time and power.
2. Way prediction to reduce hit time.
3. Pipelined cache access and Multibanked caches to increase cache bandwidth.
4. Non-blocking caches to increase cache bandwidth.
5. Critical word first and early restart to reduce miss penalty.
6. Merging write buffers to reduce miss penalty.
7. Compiler optimizations
8. Hardware prefetching to reduce miss penalty or miss rate.
9. Compiler controlled prefetching to reduce miss penalty or miss rate.
Small & simple L1, reduce hit time/power

- Direct mapped to overlap fetch with tag lookup
- Constrain associativity for power
- Currently designers favor more associativity to larger caches
Relative Access Time of Caches

![Bar chart showing relative access time in microseconds for different cache sizes and ways.](chart_url)
Energy Consumption of Caches

The chart shows the relative energy consumption per read in nano joules for different cache sizes and ways. The x-axis represents the cache size in kilobytes (KB), and the y-axis shows the relative energy consumption. The bars are color-coded for 1-way, 2-way, 4-way, and 8-way caches.
Way prediction to reduce hit time

- add bit to $k$-way set associative caches to predict the first block to search
- simulation suggests 90% effective in 2-way and 80% effective in 4-way
Pipeline/Multibank cache: increase bandwidth

- a simple pipeline: fetch and tag match
- most high speed processors have pipelined L1 caches (2-4 stages)
- multibanked for independent and concurrent access
- generally low-order interleaved (sequential interleaving)
### Multibanked Caches: Sequential Interleaving

<table>
<thead>
<tr>
<th>Block address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Non-blocking caches: increase bandwidth

- *hit under miss*: continue to deliver data while miss is processed
- *hit under multiple miss* or *miss under miss*
- design cache to handle multiple requests concurrently
- closely tied to out-of-order instruction execution
Critical word first/early restart: reduce miss penalty

- **critical word first**: on miss first deliver data than backfill the cache
- **early restart**: fill cache but deliver data to CPU as it arrives
Merging write buffers to reduce miss penalty

- in buffer used to write a cache block data
  - if write buffer has block match, merge the data in
  - if no block match, add the write data to the buffer
  - if buffer full, wait
Compiler optimizations

- Loop interchange (row → column order)
- Blocking
Hardware prefetching: reduce miss penalty/rate

- often instruction prefetching occurs in the CPU and not the cache
- prefetching uses bandwidth that might cause issues elsewhere (multicore)
Compiler controlled prefetch: reduce miss penalty/rate

- **register prefetch**: loads a value into a register
- **cache prefetch**: loads into the cache only (special instructions)
- **faulting** prefetch can trigger a Virt Mem fault; *non-faulting* Virt Mem fault turns prefetch into a no-op
- only makes sense if CPU can continue on prefetch