CMSC 611: Advanced Computer Architecture

Cache 2
Measuring Cache Performance

• To enhance cache performance, one can:
  – Reduce the miss rate (e.g. diminishing blocks collisions)
  – Reduce the miss penalty (e.g. adding multi-level caching)
  – Enhance hit access time (e.g. simple and small cache)

**CPU time** = (CPU execution cycles + Memory stall cycles) × Cycle time

**Memory stall cycles** = Read stall cycles + Write stall cycles

**Read stall cycles** = \( \frac{\text{Reads}}{\text{Program}} \) × Read miss rate × Read miss penalty

For write-through scheme:

**Write stall cycles** = \( \frac{\text{Writes}}{\text{Program}} \) × Write miss rate × Write miss penalty + Write buffer stalls

Hard to control, assume enough buffer size
Example

Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed. Assume 36% combined frequencies for load and store instructions.

Answer:

Assume number of instructions = I

Instruction miss cycles = I \times 2\% \times 40 = 0.8 \times I

Data miss cycles = I \times 36\% \times 4\% \times 40 = 0.56 \times I

Total number of memory-stall cycles = 0.8 I + 0.56 I = 1.36 I

The CPI with memory stalls = 2 + 1.36 = 3.36

\[
\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{\text{stall}} \times \text{Clock cycle}}{I \times CPI_{\text{perfect}} \times \text{Clock cycle}} = \frac{CPI_{\text{stall}}}{CPI_{\text{perfect}}} = \frac{3.36}{2}
\]

What happens if the CPU gets faster?
Classifying Cache Misses

- **Compulsory**
  - First access to a block not in cache
  - Also called cold start or first reference misses
  - (Misses in even an Infinite Cache)

- **Capacity**
  - If the cache cannot contain all needed blocks
  - Due to blocks discarded and re-retrieved
  - (Misses in Fully Associative Cache)

- **Conflict**
  - Set associative or direct mapped: too many blocks in set
  - Also called collision or interference
  - (Misses in N-way Associative Cache)
Improving Cache Performance

- Capacity misses can be damaging to the performance (excessive main memory access)
- Increasing associativity, cache size and block width can reduces misses
- Changing cache size affects both capacity and conflict misses since it spreads out references to more blocks
- Some optimization techniques that reduces miss rate also increases hit access time
Miss Rate Distribution

- Compulsory misses are small compared to other categories
- Capacity misses diminish with increased cache size
- Increasing associativity limits the placement conflicts

Based on SPEC92
Techniques for Reducing Misses

CPUtime = IC \times \left( \frac{CPI_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{Clock cycle time}

1. Reducing Misses via Larger Block Size
2. Reducing Misses via Higher Associativity
3. Reducing Misses via Victim Cache
4. Reducing Misses via Pseudo-Associativity
5. Reducing Misses by H/W Prefetching Instr. and Data
6. Reducing Misses by S/W Prefetching Data
7. Reducing Misses by Compiler Optimizations
• Larger block sizes reduces compulsory misses (principle of spatial locality)
• Conflict misses increase for larger block sizes since cache has fewer blocks
• The miss penalty usually outweighs the decrease of the miss rate making large block sizes less favored
Reduce Misses via Higher Associativity

- Greater associativity comes at the expense of larger hit access time
- Hardware complexity grows for high associativity and clock cycle increases

2:1 Cache Rule:
Miss Rate for direct mapped cache of size $N$ = Miss Rate 2-way cache size $N/2$
### Example

Assume hit time is 1 clock cycle and average miss penalty is 50 clock cycles for a direct mapped cache. The clock cycle increases by a factor of 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way associative cache. Compare the average memory access based on the previous figure miss rates.

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.65</td>
<td>6.60</td>
<td>6.22</td>
<td>5.44</td>
</tr>
<tr>
<td>2</td>
<td>5.90</td>
<td>4.90</td>
<td>4.62</td>
<td>4.09</td>
</tr>
<tr>
<td>4</td>
<td>4.60</td>
<td>3.95</td>
<td>3.57</td>
<td>3.19</td>
</tr>
<tr>
<td>8</td>
<td>3.30</td>
<td>3.00</td>
<td>2.87</td>
<td>2.59</td>
</tr>
<tr>
<td>16</td>
<td>2.45</td>
<td>2.20</td>
<td>2.12</td>
<td>2.04</td>
</tr>
<tr>
<td>32</td>
<td>2.00</td>
<td>1.80</td>
<td>1.77</td>
<td>1.79</td>
</tr>
<tr>
<td>64</td>
<td>1.70</td>
<td>1.60</td>
<td>1.57</td>
<td>1.59</td>
</tr>
<tr>
<td>128</td>
<td>1.50</td>
<td>1.45</td>
<td>1.42</td>
<td>1.44</td>
</tr>
</tbody>
</table>

A good size of direct mapped cache can be very efficient given its simplicity. High associativity becomes a negative aspect.
• Combines fast hit time of direct mapped yet still avoids conflict misses
  – Adds small fully associative cache between the direct mapped cache and memory to place data discarded from cache
  – Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
  – Technique is used in Alpha, HP machines and does not impair the clock rate
Pseudo-Associativity Mechanism

- Combine fast hit time of Direct Mapped and lower conflict misses of 2-way set associative
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit
- Simplest implementation inverts the index field MSB to find the other pseudo set
- To limit the impact of hit time variability on performance, swap block contents
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC
H/W Pre-fetching of Instructions & Data

- Hardware pre-fetches instructions and data while handing other cache misses
  - Assume pre-fetched items will be referenced shortly
- Pre-fetching relies on having extra memory bandwidth that can be used without penalty

\[
\text{Average memory access time} = \text{Hit time} + \text{Miss Rate} \times \\
(\text{Prefetch hit rate} + (1 - \text{Prefetch hit rate}) \times \text{Miss penalty})
\]

- Examples of Instruction Pre-fetching:
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in “stream buffer”
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
Software Pre-fetching Data

- Uses special instructions to pre-fetch data:
  - Load data into register (HP PA-RISC loads)
  - Cache Pre-fetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
- Special pre-fetching instructions cannot cause faults (undesired exceptions) since it is a form of speculative execution
- Makes sense if the processor can proceeds without blocking for a cache access (lock-free cache)
- Loops are typical target for pre-fetching after unrolling (miss penalty is small) or after applying software pipelining (miss penalty is large)
- Issuing Pre-fetch Instructions takes time
  - Is cost of pre-fetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

```c
for (i = 0; i < 3; i = i+1)
for (j = 0; j < 100; j = j+1)
    a[i][j] = b[j][0] * b[j+1][0];
```

```c
for (j = 0; j < 100; j = j+1)
    pre-fetch (b[i+7][0]);
    a[0][j] = b[j][0] * b[j+1][0];
for (i = 1; i < 3; i = i+1)
    pre-fetch (a[i][j+7]);
    a[i-1][j] = b[j][0] * b[j+1][0];
```