A Common Framework for Memory Hierarchies

Caching, paged virtual memory and TLBs all use the same underlying concepts

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cache</th>
<th>Paged Mem</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size, Blocks</td>
<td>1K-100K</td>
<td>2K-250K</td>
<td>32-400K</td>
</tr>
<tr>
<td>Size, Bytes</td>
<td>8KB-8MB</td>
<td>8MB-8GB</td>
<td>128B-8000B</td>
</tr>
<tr>
<td>Blk Size, B</td>
<td>4-256</td>
<td>4KB-64KB</td>
<td>4-32</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>10-100clk</td>
<td>1M-10Mclk</td>
<td>10-100clk</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>0.1%-10%</td>
<td>10^-4-10^-5%</td>
<td>0.01%-2%</td>
</tr>
</tbody>
</table>

Four Questions for Classification

• Where can a block be placed? Block placement
  – direct mapped, set associative, fully associative
• How is a block found? Block identification
  – indexing, set search, separate lookup table
• What block is replaced on a miss? Block replacement
  – LRU, Random, FIFO, MRU
• How are writes handled? Write strategy
  – write through or write back

Summary and Review

Block Placement

The extremes of cache mapping -- direct mapped and fully associative are end points on a spectrum

Blocks are assigned to a cache by directly indexing any of its n sets and matching any of the m entries of the set associatively by the tag

Indexing is "block number modulo number of sets"

12 MOD 8 = 4

Direct Mapped 4-Way Set Associative Fully Associative

Block Identification

Placement of a block whose address is 12 varies for direct, set associative, and fully associative

Block Replacement

• Replacement candidates are --
  – Any block in a fully associative cache
  – Any block of a set in set associative caches
  – The indexed block for direct mapped
• Replacement strategies --
  – Opt is best, but impossible
  – Least Recently Used (LRU) approximates Opt. Expensive
  – Random is easy, but impossible for software management
• For 2-way s.a., random has 1.1 times higher miss rate than LRU
• "Use" bit can approximate LRU

Write Strategy

• Write through simultaneously updates the cache and the lower level in the memory hierarchy on each write.
• Write back only updates the cache copy until the block is replaced, at which point the next lower level of the hierarchy is updated.
• Write through advantages --
  – Read misses are cheaper due to not waiting for write. Easier to implement, though it needs a write buffer.
• Write back advantages --
  – Multiple writes to a block require only one memory write.
  – Can utilize wider channel to lower level memory.
• Write back is always needed between memory & disk.
  – Dirty bit in page table determines if write back needed.
Mapping Choices in Hierarchy

- Tradeoff cost of miss vs cost of associativity.
- VM uses fully associative mapping
  - Reduces miss rate, because miss penalty is high
  - Mapping done in software
  - Large page size means page table size overhead is small
  - Note that page table is indexed, but full map provides fully associative placement
- Small caches (TLB) often use set associative placement
- Large caches never use fully associative placement
  - High cost and hit time penalties
  - Small performance advantage to set associative

The Three Cs

Missing in the cache can be caused by three different circumstances:

- Compulsory misses -- miss on first access
- Capacity misses -- miss due to cache not having enough blocks
- Conflict misses -- miss due to cache organization

In cache design, larger is always better ... but there are always trade-offs

Miss Rates

The Problem with Miss-rate

It doesn't tell the whole story:
Consider increasing direct-mapped cache from 32K to 64K
Miss Rate drops from 5% to 4%. If the larger cache implies a cycle time of 18ns and the smaller cache implies a cycle time of 15ns, the smaller cache machine has better performance

Postulate: CPI w/o stalls is unchanged
Memory references per instruction = 1.5
CPU Time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time

Cache Analysis, Continued

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>Memory-stall clock cycles</th>
<th>Instructions</th>
<th>Misses</th>
<th>Miss penalty</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smaller Cache</td>
<td>IC x (0.05 x 0.05 x 0.05) x IC</td>
<td>Miss rate + Data rate x Data references</td>
<td>IC x 0.05</td>
<td>0.9C</td>
<td></td>
</tr>
<tr>
<td>Larger Cache</td>
<td>IC x (0.04 x 0.04 x 0.04) x IC</td>
<td>Miss rate + Data rate x Data references</td>
<td>IC x 0.06</td>
<td>0.8C</td>
<td></td>
</tr>
</tbody>
</table>

Let IC be instructions per program

Small Cache
Memory-stall clock cycles = IC x 0.05
Absolute miss penalty = IC x 0.05
Largest cycle time = IC x 0.05 x 180/15 = .9C

Large Cache
Memory-stall clock cycles = IC x 0.04
Absolute miss penalty = IC x 0.06
Largest cycle time = IC x 0.06 x 180/18 = .8C

Cache Analysis, Continued

Memory-stall clock cycles = 0.9IC (Small) and 0.6IC (Large cache).
Substituting into the CPU time equation, letting CPI w/o stalls be C:

CPU Time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time

Small Cache
CPU Time = IC x (0.05 + 0.05 x 0.5) x 15ns + 15C x 13.5 IC

Large Cache
CPU Time = IC x (0.04 + 0.04 x 0.5) x 18ns + 18C x 10.8 IC

For C ≥ 1 the smaller cache is better