Lecture 12-13: Memory Hierarchy: Five Ways to Reduce Miss Penalty

1. Reducing Miss Penalty: Read Priority over Write on Miss

- **Write through with write buffers**
  - RAW conflicts with main memory reads on cache misses
  - If simply wait for write buffer to empty might increase read miss penalty by 50% (old MIPS 1000)
  - Check write buffer contents before read; if no conflicts, let the memory read continue

- **Write back with write buffers**
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
2. Subblock Placement to Reduce Miss Penalty

- To reduce memory required for tags
  - Reduce number of full tags and make other tags smaller
  - The resulting blocks are large
  - To reduce miss penalty
  - Don't load full block on a miss
  - Have bits per subblock to indicate valid

3. Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
- Generally useful only in large blocks
4. Non-blocking Caches to reduce stalls on misses

- *Non-blocking cache* or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss
- “hit under miss” reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
- “hit under multiple miss” or “miss under miss” may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses

Value of Hit Under Miss for SPEC

- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss
5th Miss Penalty Reduction: Second Level Cache

• L2 Equations
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]
  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} + \text{Miss Penalty}_{L2}) \]

• Definitions:
  - Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss rate\textsubscript{L2})
  - Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU
  For L2: (Miss Rate\textsubscript{L1} \times Miss Rate\textsubscript{L2})

Comparing Local and Global Miss Rates

• Global miss rate close to single level cache rate provided L2 >> L1 — any misses that are occurring in L2 would occur whether or not L1 is present.
• Don’t use local miss rate because it is a function of the miss rate of the first level cache.
• L2 not tied to CPU clock cycle.
• Size & A.M.A.T.
Reducing Misses in L2 Cache

- Since hit time is less important, make L2 caches large to avoid capacity misses
- Use Larger Block Size to reduce Compulsory Misses
- Reduce Conflict Misses via Higher Associativity

L2 cache block size & A.M.A.T.

- 512KB L2, 4 byte path to memory
Reducing Miss Penalty Summary

- Read priority over write on miss
- Subblock placement
- Early Restart and Critical Word First on miss
- Non-blocking Caches (Hit Under Miss)
- Second Level Cache

Review: Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
1. Fast Hit times via Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- Direct Mapped, on chip
- Advantage of direct mapped: Overlap tag check with reading of data to further reduce hit time

2. Fast hits by Avoiding Address Translation

- Send virtual address to cache? Called Virtually Addressed Cache or just Virtual Cache vs. Physical Cache
  - Every time process is switched logically must flush the cache; otherwise get false hits
    » Cost is time to flush + “compulsory” misses from empty cache
  - Dealing with aliases (sometimes called synonyms):
    Two different virtual addresses map to same physical address
- Solution to aliases
  - HW that guarantees that no two cache blocks have same physical address.
  - SW guarantee: lower n bits must have same address; as long as covers index field &
    direct mapped, they must be unique; called page coloring
- Solution to cache flush
  - Add process identifier tag that identifies process as well as address within process: can’t get a hit if wrong process
Avoiding Translation During Indexing: Virtually Indexed and Physically Tagged

• If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag

<table>
<thead>
<tr>
<th>Page Address</th>
<th>Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Tag</td>
<td>Index</td>
</tr>
<tr>
<td></td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

• Limits cache to page size: what if want bigger caches?
  – Higher associativity

3. Fast Hit Times Via Pipelined Writes

• Pipeline Tag Check and Update Cache as separate stages;
• Current write tag check & previous write cache update
## Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Priority to Read Misses</td>
<td>+</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Subblock Placement</td>
<td>+</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Second Level Caches</td>
<td>+</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Small &amp; Simple Caches</td>
<td>-</td>
<td>+</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Avoiding Address Translation</td>
<td>-</td>
<td>+</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pipelining Writes</td>
<td>-</td>
<td>+</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>