Memory Hierarchy—Improving Performance

Professor Alvin R. Lebeck
Computer Science 220
Fall 2001

Outline of Today’s Lecture

Review
• The Memory Hierarchy
• Associativity
• Replacement Policies
• Write Strategies

Today
• Memory Hierarchy Performance
• Improving Performance

Who Cares About the Memory Hierarchy?

• Processor Only Thus Far in Course:
  – CPU cost/performance, ISA, Pipelined Execution

The Motivation for Caches

• Motivation:
  – Large memories (DRAM) are slow
  – Small memories (SRAM) are fast

• Make the average access time small by:
  – Servicing most accesses from a small, fast memory.
• Reduce the bandwidth required of the large memory

The Principle of Locality

• The Principle of Locality:
  – Program access a relatively small portion of the address space at any instant of time.
  – Example: 90% of time in 10% of the code

• Two Different Types of Locality:
  – Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  – Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
Review: Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? (Block placement)
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? (Block identification)
  - Tag/Block
- Q3: Which block should be replaced on a miss? (Block replacement)
  - Random, LRU
- Q4: What happens on a write? (Write strategy)
  - Write Back or Write Through (with Write Buffer)

ABCs of caches

- Associativity
- Block size
- Capacity
- Number of sets \( S = \frac{C}{B} \)
  - 1-way (Direct-mapped)
    - \( A = 1, S = \frac{C}{B} \)
  - N-way set-associative
  - Fully associativity
    - \( S = 1, C = BA \)
- Know how a specific piece of data is found
  - Index, tag, block offset

Sub-block Cache (Sected)

- Sub-block:
  - Share one cache tag between all sub-blocks in a block
  - Each sub-block within a block has its own valid bit
- Miss: only the bytes in that sub-block are brought in.
  - reduces cache fill bandwidth (penalty).

Example

- Miss penalty 50 clocks
- Miss rate 2%
- Base CPI 2.0
- 1.33 references per instruction
- Compute the CPtime

\[ \text{CPtime} = IC \times (CPI_{\text{execution}} + \text{Misses per instruction} \times \text{Miss penalty}) \times \text{Clock cycle time} \]
Example 2

• Two caches: both 64KB, 32 byte blocks, miss penalty 70ns, 1.3 references per instruction, CPI 2.0 w/ perfect cache
  • direct mapped
    – Cycle time 2ns
    – Miss rate 1.4%
  • 2-way associative
    – Cycle time increases by 10%
    – Miss rate 1.0%
  • Which is better?
    – Compute average memory access time
    – Compute CPU time

Example 2 Continued

• Ave Mem Acc: Hit time + (miss rate x miss penalty)
  – 1-way: 2.0 + (0.014 x 70) = 2.98ns
  – 2-way: 2.2 + (0.010 x 70) = 2.90ns

• CPUtime = IC x CPIexec x Cycle
  – CPIexec = CPIbase + ((memacc/inst) x Miss rate x miss penalty)
  – Note: miss penalty x cycle time = 70ns
  – 1-way: IC x [(2.0 x 2.0) + (1.3x0.014x70)] = 5.27 x IC
  – 2-way: IC x [(2.0 x 2.2) + (1.3x0.010x70)] = 5.31 x IC

Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Reducing Misses

• Classifying Misses: 3 Cs
  – Compulsory—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called cold start misses or first reference misses.
  – Capacity—if the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
  – Conflict—if the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called collision misses or interference misses.

3Cs Absolute Miss Rate

2:1 Cache Rule
How Can We Reduce Misses?

- Change Cache Capacity?
- Change Block Size?
- Change Associativity?
- Change Program/Compiler?
- Which of 3Cs affected by these changes?

Example: Avg. Memory Access Time vs. Miss Rate

<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>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.46</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red means A.M.A.T. not improved by more associativity)

3. Reducing Conflict Misses via Victim Cache

- How to combine fast hit time of Direct Mapped yet still avoid conflict misses?
- Add buffer 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
4. Reducing Conflict Misses via Pseudo-Associativity

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)
- Hash-Rehash: probe once if miss, probe again

<table>
<thead>
<tr>
<th>Hit Time</th>
<th>Pseudo Hit Time</th>
<th>Miss Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time</td>
<td>Time</td>
</tr>
</tbody>
</table>

- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor

5. Reducing Misses by HW Prefetching of Instruction & Data

- E.g., Instruction Prefetching
  - 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
  - Kedem: Markov predictor

- Prefetching relies on extra memory bandwidth that can be used without penalty
- Need to worry about cache pollution

6. Reducing Misses by SW Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads) binding
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9) non-binding
  - Special prefetching instructions cannot cause faults; a form of speculative execution

- Issuing prefetch instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?

- Need to worry about cache pollution

7. Reducing Misses by Program/Compiler Optimizations

- Instructions
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

- Data
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows