G22.2243-001
High Performance Computer Architecture

Lecture 10
Memory Hierarchy Design (cont’d)

November 6, 2002
Outline

• Announcements
  – Assignments
    • #3 due on November 13th
    • #4 will be a moderate extension over Assignment 3
      – Out: November 13th, Due: November 27th
    • #5 will be an analysis experiment (no programming)
      – Out: November 27th, Due: December 18th
  – Final exam on December 11th: Room 402, WWH

• This lecture
  – Improving cache performance (cont’d)
  – Main memory organizations

[ Hennessy/Patterson CA:AQA (3rd Edition): Chapter 5]
(Review) Improving Cache Performance

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

Memory stall clock cycles = 
    (Reads x Read miss rate x Read miss penalty + 
    Writes x Write miss rate x Write miss penalty)

= Memory accesses x Miss rate x Miss penalty

- Above assumes 1-cycle to hit in cache
  - Hard to achieve in current-day processors (faster clocks, larger caches)
  - More reasonable to also include hit time in the performance equation

Average memory access time = Hit Time + Miss rate x Miss Penalty

- Small/simple caches
- Avoiding address translation
- Pipelined cache access
- Trace caches
- Larger block size
- Larger cache size
- Higher associativity
- Way prediction
- Compiler optimizations
- Multilevel caches
- Critical word first
- Read miss before write miss
- Merging write buffers
- Victim caches
- Nonblocking caches
- Hardware prefetching
- Compiler prefetching
A.1. Reducing Miss Penalty via Multilevel Caches

• **Idea**: Have multiple levels of caches
  – Tradeoff between size (cache effectiveness) and cost (access time)

• For a 2-level cache
  
  Average memory access time = Hit time (L1) + Miss rate (L1) x Miss penalty (L1)
  Miss penalty (L1) = Hit time (L2) + Miss rate (L2) x Miss penalty (L2)

• Distinguish between two kinds of miss rates
  – **Local** miss rate = Miss rate (L1) or Miss rate (L2)
  – **Global** miss rate = Number of misses/total number of memory accesses
    = Miss rate (L1), but Miss rate (L1) x Miss rate(L2)

• Example: 1000 references, 40 misses in L1 cache and 20 in L2
  – Local miss rates: 4% (L1), 50% (L2) = 20/40
  – Global miss rates: 4% (L1), 2% (L2)
  – Avg. memory access time = 1 + 4% x (10 + 50% x 100) = 3.4 cycles
Multilevel Caches (cont’d)

- Doesn’t make much sense to have L2 caches smaller than L1 caches
- L2 needs to be significantly bigger to have reasonable miss rates
  - Cost of big L2 is smaller than big L1

64 KB L1 caches for instruction, data
Unified L2 cache
A.2. Reduce Miss Penalty via Critical Word First and Early Restart

- **Idea:** 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

- **Drawbacks**
  - Generally useful only in large blocks
  - Programs exhibiting spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

- **Example:** 64B block, 11 cycles to get critical 8B, plus 2 cycles/8B
  - Average miss penalty: 11 cycles
  - Time to load entire block: $11 + (8-1) \times 2 = 25$ cycles
A.3. Reducing Miss Penalty by giving Reads Priority over Writes on Misses

- Write buffers ensure that writes to memory do not stall the processor
- On the other hand, processor is blocked till read returns

Solution: Give read misses priority

Challenges
- Write-through with write buffers may result in RAW conflicts
  - Solution 1: Wait for write buffer to empty (not great)
  - Solution 2: Check write buffer contents before read; if no conflicts, let the memory access continue
    - Why not just return the value in the write buffer?
- Write-back caches: Read miss may require replacing a dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Better alternative: 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 read is done
A.4. Reducing Miss Penalty using Merging Write Buffers

- Normal mode of operation of a write buffer
  - Absorb write from CPU, commit it to memory in the background

- Problem (particularly in write-through caches)
  - Small write-buffers may end up stalling processor if they fill up
  - Processor needs to wait till write committed to memory

- Solution: Merge cache-block entries in the write buffer
  - Multiword writes are usually faster than writes performed one at a time
A.5. Reducing Miss Penalty via a “Victim Cache”

- How to combine the fast hit time of direct-mapped caches, yet still avoid conflict misses?
- Remember what was recently discarded, just in case it is needed again
  - Jouppi [1990]: 4-entry victim cache reduced conflict misses by 20% - 95% for a 4 KB direct mapped data cache
  - Used in Alpha, HP machines

![Diagram of a cache system with victim cache](image)
B. Reducing Cache Misses

Classifying Misses: 3 Cs

• **Compulsory** (Also called cold start or first reference misses)
  - The first access to a block is not in the cache, so the block must be brought into the cache.
  (Misses in even an Infinite Cache)

• **Capacity**
  - The cache may not contain all blocks needed during program execution, so misses will occur due to blocks being discarded and later retrieved
  (Misses in Fully Associative Size X Cache)

• **Conflict** (Also called collision or interference misses)
  - Additional misses that occur because another block is occupying cache (the rest of the cache might be unused)
  (Misses in N-way Associative, Size X Cache)
3Cs Absolute Miss Rate (SPEC92)
2:1 Cache Rule of Thumb

- Miss rate of a 1-way associative cache of size X ~
  Miss rate of a 2-way associative cache of size X/2
3Cs Relative Miss Rate

- Assumes fixed block size for each size cache
How Can We Reduce Misses?

• 3 Cs: Compulsory, Capacity, Conflict

• If we assume that total cache size is not changed, what happens if we

1. Change block size
Which of 3Cs is obviously affected?

2. Change associativity
Which of 3Cs is obviously affected?

3. Change compiler
Which of 3Cs is obviously affected?
B.1. Reducing Miss Rate via Larger Block Sizes

- Small blocks: Data accesses spread over multiple blocks
- Large blocks: Not all the data is useful, but displaces useful data

![Graph showing the relationship between block size and miss rate]
B.2. Reducing Miss Rate via Higher Associativity

- 2:1 Cache Rule
  - Miss Rate of a direct-mapped cache size of size N ~
  - Miss Rate of a 2-way cache of size N/2

- Is this actually the case?
  - *Issue*: Increase in clock cycle time (CCT) may diminish benefits

- Average memory access time for SPEC92 vs. associativity
  - CCT = 1.0 for 1-way, 1.36 for 2-way, 1.44 for 4-way, 1.52 for 8-way

<table>
<thead>
<tr>
<th>Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.44</td>
<td>3.25</td>
<td>3.22</td>
<td>3.28</td>
</tr>
<tr>
<td>8</td>
<td>2.69</td>
<td>2.58</td>
<td>2.55</td>
<td>2.62</td>
</tr>
<tr>
<td>16</td>
<td>2.23</td>
<td>2.40</td>
<td>2.46</td>
<td>2.53</td>
</tr>
<tr>
<td>32</td>
<td>2.06</td>
<td>2.30</td>
<td>2.37</td>
<td>2.45</td>
</tr>
<tr>
<td>64</td>
<td>1.92</td>
<td>2.14</td>
<td>2.18</td>
<td>2.25</td>
</tr>
<tr>
<td>128</td>
<td>1.52</td>
<td>1.86</td>
<td>1.92</td>
<td>2.00</td>
</tr>
<tr>
<td>256</td>
<td>1.32</td>
<td>1.66</td>
<td>1.74</td>
<td>1.82</td>
</tr>
<tr>
<td>512</td>
<td>1.20</td>
<td>1.55</td>
<td>1.59</td>
<td>1.66</td>
</tr>
</tbody>
</table>
B.3. Reducing Miss Rate via Way Prediction and Pseudoassociativity

- How to combine fast hit time of direct-mapped caches with the lower conflict misses of set-associative caches?
  - Previously looked at Victim Caches

- **Way prediction**: Predict which block in a set is likely to be accessed by the next memory access hitting this set
  - Tag comparison *only* with this block (cheaper as opposed to with all)
    - Higher cost to check non-predicted blocks
  - Used in Alpha 21264 (1-cycle if correct prediction (85%), 3-cycles o.w.)

- **Pseudoassociative** or **Column associative**
  - Access proceeds as in direct-mapped cache
  - On a miss, check another location (“pseudoset”) before going to memory
    - Counts as a “slower hit”
    - Block allocation follows similar pattern, so danger of this degrading performance
  - Used in MIPS R10000 L2 cache, similar in UltraSPARC
B.4. Reducing Miss Rate by Compiler Optimizations

- Compiler optimizations can help reduce both instruction and data cache misses (for a fixed cache organization)

  - **Instruction misses**
    - **Reorder procedures** in memory so as to reduce conflict misses
      - Ensure that procedures used frequently do not map to same blocks/sets
      - Conflicts determined by profiling
      - Reduced I-cache misses by 75% in an 8KB cache (McFarling 1989)
    - **Cache-line alignment** of basic blocks
      - Decreases likelihood of cache miss on sequential code

  - **Data misses**
    - Several optimizations that reorder data access patterns
    - Two examples
      - Loop interchange
      - Blocking
Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
      \[x[i][j] = 2 * x[i][j];\]

/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
      \[x[i][j] = 2 * x[i][j];\]

- “After” version accesses memory sequentially instead of in strides of 100 words
  - Improved spatial locality: use all of the words in fetched blocks
/* Before */
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
        { r = 0;
            for (k = 0; k < N; k = k + 1){
                r = r + y[i][k]*z[k][j];
            }
            x[i][j] = r;
        };

Capacity misses depend on N, cache size
if all three matrices fit and there are
no conflict misses, best performance
if cache can hold one NxN matrix and
one row of N elements, then y and z can
be in the cache
else, misses for both y and z
worst case: 2N^3 + N^2 misses
Blocking Example (cont’d)

/* After */
for (jj = 0; jj < N; jj = jj+.blocks)
for (kk = 0; kk < N; kk = kk+blocks)
for (i = 0; i < N; i = i+1)
for (j = jj; j < min(jj+blocks-1, N); j = j+1)
  { r = 0;
    for (k = kk; k<min(kk+blocks-1, N); k = k+1) {
      r = r + y[i][k]*z[k][j];};
    x[i][j] = x[i][j] + r;
  }

Blocking factor: compute in blocks of BxB
B chosen such that 1 row of B and 1 BxB matrix can fit in the cache. This ensures that y and z blocks are resident

Capacity misses:
2N^3/B + N^2
Reducing Conflict Misses by Blocking

- Conflict misses in set-associative caches vs. blocking size
  - Lam et al [1991]: A blocking factor of 24 had 1/5\textsuperscript{th} the misses of 48 despite both fitting in the cache
C. Using Parallelism to Reduce Miss Penalty/Rate

- **Idea:** Permit multiple “outstanding” memory operations
  - Can overlap memory access latencies
  - Can benefit from activity done on behalf of other operations

Three commonly-employed schemes

- Non-blocking caches
- Hardware prefetching
- Software prefetching
C.1. Non-blocking Caches to Reduce Stalls on Misses

- Decoupled instruction and data caches allow CPU to continue fetching instructions while waiting on a data cache miss
  - L1 cache misses can be tolerated by superscalar out-of-order machines

- Non-blocking or lockup-free caches allow data cache to continue to supply cache hits during a miss
  - Requires out-of-order execution CPU

- “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests

- “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
  - Typically also requires multiple memory banks
  - Pentium Pro allows 4 outstanding memory misses
Value of Hit-Under-Miss for SPEC92
8KB direct-mapped cache, 32B blocks, 16-cycle penalty

Floating-point

Percentage of the average memory stall time

Hit under 1 miss

Hit under 2 misses

Hit under 64 misses

81%

78%

76%

51%

39%