Lecture 14: Caches and Cache Performance

- Last Time
  - Started caches

- Today
  - Improving cache performance

Questions of Cache Design

- Where can a block be placed?
- How is a block found?
- Which block should be replaced on a miss?
- What happens on a write?
Block Placement

- Mapping function from Big Memory to Small memory
- On block-by-block basis
  - Direct Mapped: 1 place
  - Fully Associative: Anywhere
  - Set Associative: Subset of cache
- Use address to do mapping and lookup

Which Block Should Be Replaced on Miss?

- Direct Mapped
  - Choice is easy - only one option

- Associative
  - Randomly select block in set to replace
  - FIFO
  - Least-Recently used (LRU)

- Implementing LRU
  - 2-way set-associative
  - >2 way set-associative
What Happens on a Store?

- Need to keep cache consistent with main memory
  - Reads are easy - no modifications
  - Writes - when do we update main memory

- Write-Through
  - On cache write - always update main memory as well
  - Use a write buffer to stockpile writes to main memory for speed

- Write-Back
  - On cache write - remember that block is modified (dirty bit)
  - Update main memory when dirty block is replaced
  - Sometimes need to flush cache (I/O, multiprocessing)

BUT: What if Store Causes Miss!

- Write-Allocate
  - Bring written block into cache
  - Update word in block
  - Anticipate further use of block

- No-write Allocate
  - Main memory is updated
  - Cache contents unmodified
How Do We Improve Cache Performance?

\[ T_{\text{access}} = \tau_{\text{hit}} + \tau_{\text{miss}} \cdot \text{penalty}_{\text{miss}} \]

- Reduce hit time
- Reduce miss rate
- Reduce miss penalty

Impact of Cache on Performance

\[ CPU_{\text{time}} = IC \cdot (CPI_{\text{execution}} + p_{\text{miss}} \cdot \text{penalty}_{\text{miss}} \cdot \frac{\text{MemAccesses}_{\text{instruction}}}{\text{instruction}}) \cdot \text{CCT} \]

- Assume memory access time = 100 cycles
- What happens when we go from perfect cache to one with 2% miss rate?
- What about if we eliminate the cache?
Reducing Hit Time

- O1: Keep the caches small and simple
- O2: Way prediction
  - Pentium4, Alpha 21264
- O4: Pipelined cache access (actually increases BW, not hit time)
- O6: Multi-banked caches (at potentially all levels)
  - Keep banks small (faster)
  - Increase BW (fewer ports per bank, faster)
- O3: Trace caches - for instruction caching
- Avoid virtual to physical translation before accessing cache

Miss Classifications

- Compulsory misses
  - First time data is accessed

- Capacity misses
  - Working set larger than cache size

- Conflict misses
  - One set fills up, but room in other sets
Reducing Miss Rate

- Increase block size...but within limits
  - Reduce compulsory/capacity/conflict misses
- Increase cache capacity
  - Reduce capacity and conflict misses
- Increase associativity
  - Reduce conflict misses
- Quasi-increases in associativity
  - Multiple places to look, but don’t look everywhere in parallel
- O10: Hardware prefetching
  - Reduce all types of misses
  - H/W and/or S/W prefetch
- O9: Compiler optimizations
  - Loop reordering: spatial locality
  - Cache blocking: temporal locality

Reducing Miss Rate: Increase Block Size

- Fetch more data with each cache miss
  - 16 bytes ⇒ 64, 128, 256 bytes!
  - Works because of Locality (spatial)
Reducing Miss Rate: Increase Associativity

- Reduce conflict misses
- Rules of thumb
  - 8-way = fully associative
  - Direct mapped size N = 2-way set associative size N/2

- But!
  - Size N associative is larger than Size N direct mapped
  - Associative typically slower than direct mapped (t_{hit} larger)

Reducing Miss Rate: Compiler Optimization (1)

- Maximize cache reuse

  - `for (j = 0; j < 100; j++)`
    `for ( i = 0; i < 5000; i++)`
    `x[i][j] += 2*x[i][j]`

  - Stride = ?

  - `for (i = 0; i < 5000; i++)`
    `for ( j = 0; j < 100; j++)`
    `x[i][j] += 2*x[i][j]`

  - Loop interchange
Reducing Miss Rate: Compiler Optimization (2)  
Blocking

for (i = 0; i < N; i++)
    for (j = 0; j < N; j++) {
        r = 0;
        for (k = 0; K < N; k++) {
            r += y[i][k] * z[k][j];
        }
        x[i][j] = r;
    }

• Before blocking
  - Poor spatial locality
  - Capacity misses

• After blocking
  - Better reuse in cache
  - Lower miss rate

© 2003 Elsevier Science (USA). All rights reserved.
Reducing Miss Rate: Prefetching

- Fetching Data that you will probably need

- Instructions
  - Alpha 21064 on cache miss
    - Fetches requested block into instruction stream buffer
    - Fetches next sequential block into cache

- Data
  - Automatically fetch data into cache (spatial locality)
  - Issues?

- Compiler controlled prefetching
  - Inserts prefetching instructions to fetch data for later use
  - Registers or cache

Reducing Miss Penalty

- Add more levels of cache
- O5: Lockup free caches
  - Multiple pending misses, miss latency overlapped
- O7: Deliver critical word to processor first, before entire block loaded into cache
- O8: Merge writes to same line if found in write buffer
- Give priority to reads over writes
- Victim caches
Reducing Miss Penalty: Use a “Victim” Cache

- Small cache (< 8 entries)
  - Jouppi 1990
  - Accessed in parallel with main cache
  - Captures conflict misses

<table>
<thead>
<tr>
<th></th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reduce Miss Penalty: More Cache Levels

- Average access time = HitTime\(_{L1}\) + MissRate\(_{L1}\) * MissPenalty\(_{L1}\)
- MissPenalty\(_{L1}\) = HitTime\(_{L2}\) + MissRate\(_{L2}\) * MissPenalty\(_{L2}\)
- etc.
- Size/Associativity of higher level caches?
Reduce Miss Penalty: Transfer Time

- We can increase the width of memory and the bus in order to decrease access times and transfer times to move things into cache.
- We can have memory “banks” that allow us to read or write multiple words in 1 access time: called *interleaving* (don’t need to change size of bus).

Reduce Miss Penalty: Deliver Critical word first

- Only need one word from block immediately
- Don’t write entire word into cache first
  - Fetch word 2 first (deliver to CPU)
  - Fetch order: 2 3 0 1
Reduce Miss Penalty: Read Misses First

- Let reads pass writes in Write buffer

SW 512(R0), R3
LW R1, 1024(R0)
LW R2, 512(R0)

Reduce Miss Penalty: Lockup Free Cache

- Let cache continue to function while miss is being serviced

LW R1, 1024(R0) ← MISS
LW R2, 512(R0)
Questions to think about

- As the block size goes up, what happens to the miss rate?
- ... what happens to the miss penalty?
- ... what happens to hit time?
- As the associativity goes up, what happens to the miss rate?
- ... what happens to the hit time?

Summary

- Cache system optimization
  - Need to keep technology constraints in mind
  - Consider also system constraints and opportunities
    - Compiler optimization
    - Memory ordering
- Next time
  - Further cache optimizations