Suggested Readings

- Readings
  - H&P: Chapter 5.3

Lecture 20
Caches: Improving Hit Time, Miss Rate, and Miss Penalty

Cache misses and the architect

- What can we do about the 3 kinds of cache misses?
  - Compulsory, capacity, and conflict...

- Can avoid conflict misses w/fully associative cache
  - But fully associative caches mean expensive HW, possibly slower clock rates, and other bad stuff

- Can avoid capacity misses by making cache bigger – small caches can lead to thrashing
  - W/thrashing, data moves between 2 levels of memory hierarchy very frequently – can really slow down perf.
Addressing Miss Rates

(1) Larger cache block size

- Easiest way to reduce miss rate is to increase cache block size
  - This will help eliminate what kind of misses?
- Helps improve miss rate b/c of principle of locality:
  - Temporal locality says that if something is accessed once, it'll probably be accessed again soon
  - Spatial locality says that if something is accessed, something nearby it will probably be accessed
    - Larger block sizes help with spatial locality

- Be careful though!
  - Larger block sizes can increase miss penalty!
    - Generally, larger blocks reduce # of total blocks in cache

(1) Larger cache block size (example)

- Assume that to access lower-level of memory hierarchy you:
  - Incur a 40 clock cycle overhead
  - Get 16 bytes of data every 2 clock cycles
- i.e. get 16 bytes in 42 clock cycles, 32 in 44, etc…
- Using data below, which block size has minimum average memory access time?

<table>
<thead>
<tr>
<th>Block Size</th>
<th>1K</th>
<th>4K</th>
<th>16K</th>
<th>64K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss Rate</td>
<td>15.05%</td>
<td>8.57%</td>
<td>3.94%</td>
<td>2.04%</td>
<td>1.09%</td>
</tr>
<tr>
<td>32</td>
<td>13.34%</td>
<td>7.24%</td>
<td>2.87%</td>
<td>1.35%</td>
<td>0.70%</td>
</tr>
<tr>
<td>64</td>
<td>13.76%</td>
<td>7.00%</td>
<td>2.64%</td>
<td>1.06%</td>
<td>0.51%</td>
</tr>
<tr>
<td>128</td>
<td>16.64%</td>
<td>7.78%</td>
<td>2.77%</td>
<td>1.02%</td>
<td>0.49%</td>
</tr>
<tr>
<td>256</td>
<td>22.01%</td>
<td>9.51%</td>
<td>3.29%</td>
<td>1.15%</td>
<td>0.49%</td>
</tr>
</tbody>
</table>
(1) Larger cache block size (cont.)

- Recall that Average memory access time =
  - Hit time + Miss rate X Miss penalty
- Assume a cache hit otherwise takes 1 clock cycle – independent of block size
- So, for a 16-byte block in a 1-KB cache...
  - Average memory access time =
    - 1 + (15.05% X 42) = 7.321 clock cycles
- And for a 256-byte block in a 256-KB cache...
  - Average memory access time =
    - 1 + (0.49% X 72) = 1.353 clock cycles

- Rest of the data is included on next slide...

(1) Larger cache block size (cont.)

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Miss Penalty</th>
<th>1K</th>
<th>4K</th>
<th>16K</th>
<th>64K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>42</td>
<td>7.321</td>
<td>4.599</td>
<td>2.655</td>
<td>1.857</td>
<td>1.485</td>
</tr>
<tr>
<td>32</td>
<td>44</td>
<td>6.870</td>
<td>4.186</td>
<td>2.263</td>
<td>1.594</td>
<td>1.308</td>
</tr>
<tr>
<td>64</td>
<td>48</td>
<td>7.605</td>
<td>4.360</td>
<td>2.267</td>
<td>1.509</td>
<td>1.245</td>
</tr>
<tr>
<td>128</td>
<td>56</td>
<td>10.318</td>
<td>5.357</td>
<td>2.551</td>
<td>1.571</td>
<td>1.274</td>
</tr>
<tr>
<td>256</td>
<td>72</td>
<td>16.847</td>
<td>7.847</td>
<td>3.369</td>
<td>1.828</td>
<td>1.353</td>
</tr>
</tbody>
</table>

Red entries are lowest average AMAT for a particular configuration

Note: Data for cache sizes in units of “clock cycles”

(1) Larger cache block sizes (wrap-up)

- We want to minimize cache miss rate & cache miss penalty at same time!

- Selection of block size depends on latency and bandwidth of lower-level memory:
  - High latency, high bandwidth encourage large block size
    - Cache gets many more bytes per miss for a small increase in miss penalty
    - (or ... if it takes a long time to process miss, bring in as much info. as you can when you do miss)
  - Low latency, low bandwidth encourage small block size
    - If it doesn’t take long to process miss, but bandwidth small, use small block sizes
      - (so miss penalty doesn’t go up because of transfer time)
    - Larger # of small blocks may reduce conflict misses
(2) Higher associativity

- Higher associativity can improve cache miss rates...

- Note that an 8-way set associative cache is...
  - ...essentially a fully-associative cache

- But, diminishing returns set in sooner or later...
  - Greater associativity can cause increased hit time

Addressing Miss Penalties

2nd-level caches

- This technique focuses on cache/main memory interface

- Processor/memory performance gap makes us consider:
  - If we should make caches faster to keep pace with CPUs
  - If we should make caches larger to overcome widening gap between CPU and main memory

- One solution is to do both:
  - Add another level of cache (L2) between the 1st level cache (L1) and main memory
    - Ideally L1 will be fast enough to match the speed of the CPU while L2 will be large enough to reduce the penalty of going to main memory

Second-level caches

- This will of course introduce a new definition for average memory access time:
  - Hit time_{L1} + Miss Rate_{L1} * Miss Penalty_{L1}
  - Where, Miss Penalty_{L1} =
    - Hit Time_{L2} + Miss Rate_{L2} * Miss Penalty_{L2}
    - So 2nd level miss rate measure from 1st level cache misses...

- A few definitions to avoid confusion:
  - Local miss rate:
    - # of misses in the cache divided by total # of memory accesses to the cache – specifically Miss Rate_{L2}
  - Global miss rate:
    - # of misses in the cache divided by total # of memory accesses generated by the CPU – specifically:
      - Miss Rate_{L1} * Miss Rate_{L2}
Second-level caches

• Example:
  – In 1000 memory references there are 40 misses in the L1 cache and 20 misses in the L2 cache. What are the various miss rates?
    – Miss Rate L1 (local or global): \( \frac{40}{1000} = 4\% \)
    – Miss Rate L2 (local): \( \frac{20}{40} = 50\% \)
    – Miss Rate L2 (global): \( \frac{20}{1000} = 2\% \)

• Local cache rate not good measure of secondary caches – its a function of L1 miss rate
  • Which can vary by changing the L1 cache
  • Use global cache miss rate to evaluating 2nd level caches!

Second-level caches (example)

• Given the following data...
  – 2-way set associativity increases hit time by 10% of a CPU clock cycle
  – Hit time for L2 direct mapped cache is: 10 clock cycles
  – Local miss rate for L2 direct mapped cache is: 25%
  – Local miss rate for L2 2-way set associative cache is: 20%
  – Miss penalty for the L2 cache is: 50 clock cycles

• What is the impact of using a 2-way set associative cache on our miss penalty?

Second-level caches

• 2nd level caches are usually BIG!
  – Usually L1 is a subset of L2
  – Should have few capacity misses in L2 cache
    • Only worry about compulsory and conflict for optimizations...

Second-level caches (example)

• Miss penalty / \( \text{AMAT}_{\text{Direct mapped L2}} \) =
  – \( 10 + 25\% * 50 = 22.5 \) clock cycles

• Adding the cost of associativity increases the hit cost by only 0.1 clock cycles
  – Thus, Miss penalty / \( \text{AMAT}_{\text{2-way set associative L2}} \) =
    • \( 10.1 + 20\% * 50 = 20.1 \) clock cycles
Second-level caches (example)

• However, we can’t have a fraction for a number of clock cycles (i.e. 10.1 not possible!)

• We’ll either need to round up to 11 or optimize some more to get it down to 10. So...
  – 10 + 20% * 50 = 20.0 clock cycles or
  – 11 + 20% * 50 = 21.0 clock cycles
    • (both better than 22.5)

Addressing Hit Time

Reducing the hit time

• Again, recall our average memory access time equation:
  – Hit time + Miss Rate * Miss Penalty
  – We’ve talked about reducing the Miss Rate and the Miss Penalty – Hit time can also be a big component...
• On many machines cache accesses can affect the clock cycle time – so making this small is a good thing

Small and simple caches

• Why is this good?
  – Generally, smaller hardware is faster – so a small cache should help the hit time...
  – Direct mapping also falls under the category of “simple”
    • Relates to point above as well – you can check tag and read data at the same time!
A cache example:

- We want to compare the following:
  - A 16-KB data cache & a 16-KB instruction cache versus a 32-KB unified cache

- Assume a hit takes 1 clock cycle to process
- Miss penalty = 50 clock cycles
- In unified cache, load or store hit takes 1 extra clock cycle b/c having only 1 cache port = a structural hazard
- 75% of accesses are instruction references
- What’s avg. memory access time in each case?

### Size | Inst. Cache | Data Cache | Unified Cache (Miss Rates)
--- | --- | --- | ---
16 KB | 0.64% | 6.47% | 2.87%
32 KB | 0.15% | 4.82% | 1.99%

### A cache example continued…

- **1st**, let’s determine overall miss rate for split caches:
  - (75% x 0.64%) + (25% x 6.47%) = 2.10%
  - This compares to the unified cache miss rate of 1.99%

- We’ll use average memory access time formula from a few slides ago but break it up into instruction & data references

- **Average memory access time – split cache** =
  - 75% x (1 + 0.64% x 50) + 25% x (1 + 6.47% x 50)
  - (75% x 1.32) + (25% x 4.235) = 2.05 cycles

- **Average memory access time – unified cache** =
  - 75% x (1 + 1.99% x 50) + 25% x (1 + 1 + 1.99% x 50)
  - (75% x 1.995) + (25% x 2.995) = 2.24 cycles

- Despite higher miss rate, access time faster for split cache!