IC220 Set #17:
Caching Finale and Virtual Reality
(Chapter 5)

Cache Performance

- Simplified model:
  \[
  \text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}
  = \frac{\text{execTime}}{\text{cycle time}} + \frac{\text{stallTime}}{\text{cycle time}}
  \]

- Two typical ways of improving performance:
  - decreasing the miss rate
  - decreasing the miss penalty

  \[
  \text{MissPenalty} = \frac{\text{MemoryAccesses}}{\text{Program}} \times \text{MissRate} \times \text{MissPenalty}
  \]

  \[
  \text{MissPenalty} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{MissPenalty}
  \]

- What happens if we increase block size?

Performance Example

- Suppose processor has a CPI of 1.5 given a perfect cache. If there are 1.2 memory accesses per instruction, a miss penalty of 20 cycles, and a miss rate of 10%, what is the effective CPI with the real cache?
Split Caches

• Instructions and data have different properties
  – May benefit from different cache organizations (block size, assoc…)

• Why else might we want to do this?

Cache Complexities

• Not always easy to understand implications of caches:

[Diagrams showing theoretical vs. observed behavior of Radix sort vs. Quicksort]

Program Design for Caches – Example 1

• Option #1
  ```c
  for (j = 0; j < 20; j++)
  for (i = 0; i < 200; i++)
  x[i][j] = x[i][j] + 1;
  ```

• Option #2
  ```c
  for (i = 0; i < 200; i++)
  for (j = 0; j < 20; j++)
  x[i][j] = x[i][j] + 1;
  ```

Cache Complexities

• Here is why:

  - Memory system performance is often critical factor
    – multilevel caches, pipelined processors, make it harder to predict outcomes
    – Compiler optimizations to increase locality sometimes hurt ILP

  - Difficult to predict best algorithm: need experimental data
Program Design for Caches – Example 2

- Why might this code be problematic?
  ```c
  int A[1024][1024];
  int B[1024][1024];
  for (i = 0; i < 1024; i++)
      for (j = 0; j < 1024; j++)
          A[i][j] += B[i][j];
  ```

- How to fix it?
Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)
- Advantages:
  - Illusion of having more physical memory
  - Program relocation
  - Protection
- Note that main point is caching of disk in main memory but will affect all our memory references!

Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty (slow disk), thus
    - pages should be fairly
    - Replacement strategy:
      - can handle the faults in software instead of hardware
- Writeback or write-through?

Address Translation

Terminology:
- Cache block
- Cache miss
- Cache tag
- Byte offset

Page Tables
Example – Address Translation Part 1

- Our virtual memory system has:
  - 32 bit virtual addresses
  - 28 bit physical addresses
  - 4096 byte page sizes

- How to split a virtual address?
  - Virtual page #
  - Page offset

- What will the physical address look like?
  - Physical page #
  - Page offset

- How many entries in the page table?

Example – Address Translation Part 2

<table>
<thead>
<tr>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translate the following addresses:</td>
</tr>
<tr>
<td>1. C0001560</td>
</tr>
<tr>
<td>2. C0006123</td>
</tr>
<tr>
<td>3. C0002450</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Page or Disk Block #</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0000</td>
<td>1</td>
</tr>
<tr>
<td>C0001</td>
<td>1</td>
</tr>
<tr>
<td>C0002</td>
<td>0</td>
</tr>
<tr>
<td>C0003</td>
<td>1</td>
</tr>
<tr>
<td>C0004</td>
<td>1</td>
</tr>
<tr>
<td>C0005</td>
<td>0</td>
</tr>
<tr>
<td>C0006</td>
<td>1</td>
</tr>
</tbody>
</table>

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer

Protection and Address Spaces

- Every program has its own "address space"
  - Program A's address 0xc000 0200 not same as program B's
  - OS maps every virtual address to distinct physical addresses

- How do we make this work?
  - Page tables –
  - TLB –

- Can program A access data from program B? Yes, if...
  1. OS can map different virtual page #'s to same physical page #'s
  2. Program A has read or write access to the page
  3. OS uses supervisor/kernel protection to prevent user programs from modifying page table/TLB

Typical values: 16-512 PTEs (page table entries), miss-rate: .01% - 1%
miss-penalty: 10 – 100 cycles
Integrating Virtual Memory, TLBs, and Caches

Modern Systems

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>IBM System/370</th>
<th>Intel Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>8 KB instruction and data caches</td>
<td>32 KB instruction and data caches</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>32 KB write-back cache</td>
<td>32 KB write-back cache</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>256 KB write-back cache</td>
<td>256 KB write-back cache</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>256 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td>L4 cache organization</td>
<td>2 GB write-back cache</td>
<td>2 GB write-back cache</td>
</tr>
<tr>
<td>L4 cache size</td>
<td>2 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>L5 cache organization</td>
<td>32 GB write-back cache</td>
<td>32 GB write-back cache</td>
</tr>
<tr>
<td>L5 cache size</td>
<td>32 GB</td>
<td>32 GB</td>
</tr>
</tbody>
</table>

TLBs and Caches

What happens after translation?

Concluding Remarks

• Fast memories are small, large memories are slow
  – We really want fast, large memories
  – Caching gives this illusion
• Principle of locality
  – Programs use a small part of their memory space frequently
• Memory hierarchy
  – L1 cache ⇔ L2 cache ⇔ ... ⇔ DRAM memory ⇔ disk
• Memory system design is critical for multiprocessors