IC220 Set #18: Caching Finale and Virtual Reality (Chapter 7)

Cache Performance

- Simplified model:
  \[
  \text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}
  = \text{execTime} + \text{stallTime}
  \]
  \[
  \text{stall cycles} = \frac{\text{MemoryAccesses} \cdot \text{MissRate} \cdot \text{MissPenalty}}{\text{Program Instructions}}
  \text{or}
  = \frac{\text{Misses} \cdot \text{MissPenalty}}{\text{Program Instruction}}
  \]

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

  *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?

*Add associativity?*
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:

Program Design for Caches – Example 1

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

- Option #2
  ```
  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 1

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

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

<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Page offset</th>
<th>Memory address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27</td>
<td>15 14 13 12 11 10 9 8</td>
<td>3 2 1 0</td>
</tr>
</tbody>
</table>

Virtual memory summary (part 2)

Data access without virtual memory:

```
Disk
Memory
```

Data access with virtual memory:

```
Disk
Memory
```

“all problems in Computer Science can be solved by another level of indirection”
-- Butler Lampson
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

Virtual address

Translation

Physical address

Page Tables

Virtual page number

Page table

Physical page or disk address

Physical memory

Valid

Disk storage
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?
  
<table>
<thead>
<tr>
<th>Virtual page #</th>
<th>Page offset</th>
</tr>
</thead>
</table>

- What will the physical address look like?

<table>
<thead>
<tr>
<th>Physical page #</th>
<th>Page offset</th>
</tr>
</thead>
</table>

- How many entries in the page table?

Example – Address Translation Part 2

<table>
<thead>
<tr>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page or Disk Block #</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>C0000</td>
</tr>
<tr>
<td>C0001</td>
</tr>
<tr>
<td>C0002</td>
</tr>
<tr>
<td>C0003</td>
</tr>
<tr>
<td>C0004</td>
</tr>
<tr>
<td>C0005</td>
</tr>
<tr>
<td>C0006</td>
</tr>
</tbody>
</table>

Translate the following addresses:
1. C0001560
2. C0006123
3. C0002450

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer

![Diagram of hardware systems] (Diagram not available)

Typical values: 16-512 entries, miss-rate: .01% - 1%, miss-penalty: 10 – 100 cycles

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
     - So A’s 0xc000 0200 = B’s 0xb320 0200
  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
Integrating Virtual Memory, TLBs, and Caches

(Figure 7.25)

Modern Systems

- Things are getting complicated!

<table>
<thead>
<tr>
<th>Item</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TLBs and Caches

What happens after translation?

Virtual address

<table>
<thead>
<tr>
<th>31 30 29 28 27</th>
<th>15 14 13 12 11 10 9 8</th>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page offset</td>
<td>Virtual page number</td>
<td></td>
</tr>
</tbody>
</table>

Physical address

Translate

<table>
<thead>
<tr>
<th>29 28 27</th>
<th>15 14 13 12 11 10 9 8</th>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page offset</td>
<td>Physical page number</td>
<td></td>
</tr>
</tbody>
</table>

Modern Systems

- Processor speeds continue to increase very fast
  — much faster than either DRAM or disk access times

Some Issues

- Design challenge: dealing with this growing disparity
  - Prefetching? 3rd level caches and more? Memory design?