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

Cache Performance

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

- Stall cycles:
  \[
  \text{stall cycles} = \frac{\text{MemoryAccesses}}{\text{Program}} \times \text{MissRate} \times \text{MissPenalty}
  \]
  (or)
  \[
  \text{stall cycles} = \frac{\text{Instructions}}{\text{Program}} \times \text{Misses} \times \text{MissPenalty}
  \]

- 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?
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?

ICache (L1)  DCache (L1)
          L1
L2 Cache   L2 Cache
Main memory Main memory

Cache Complexities

- Not always easy to understand implications of caches:

[Graphs showing theoretical and 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;
  ```

- Program Design for Caches

- 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?
  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

Virtual memory summary (part 1)

Data access without virtual memory:

Virtual memory summary (part 2)

Data access with virtual memory:
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!

Address Translation

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

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?

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?

<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>Physical Page or Disk Block #</th>
<th>Valid?</th>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0000</td>
<td>1</td>
<td>A020</td>
</tr>
<tr>
<td>C0001</td>
<td>1</td>
<td>A200</td>
</tr>
<tr>
<td>C0002</td>
<td>0</td>
<td>FB00</td>
</tr>
<tr>
<td>C0003</td>
<td>1</td>
<td>8003</td>
</tr>
<tr>
<td>C0004</td>
<td>1</td>
<td>7290</td>
</tr>
<tr>
<td>C0005</td>
<td>0</td>
<td>5600</td>
</tr>
<tr>
<td>C0006</td>
<td>1</td>
<td>5F0C</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](image)

  Typical values: 16-512 entries,
  miss-rate: 0.01% - 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

What happens after translation?

Virtual address

Virtual page number Page offset

Translation

Physical address

Physical page number Page offset

Cache

TLBs and Caches

Modern Systems

Concluding Remarks

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Instr Instructions</th>
<th>Data Instructions</th>
<th>LRU Options &amp; Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>Split instruction and data caches</td>
<td>Split instruction and data caches</td>
<td></td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KB each for instructions/data per core</td>
<td>64 KB each for instructions/data per core</td>
<td></td>
</tr>
<tr>
<td>L1 cache associativity</td>
<td>4-way (1), 8-way (D) set associative</td>
<td>2-way set associative</td>
<td></td>
</tr>
<tr>
<td>L1 replacement</td>
<td>Approximated LRU replacement</td>
<td>LRU replacement</td>
<td></td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
<td></td>
</tr>
<tr>
<td>L1 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
<td></td>
</tr>
<tr>
<td>L1 hit time (Instruction)</td>
<td>Not Available</td>
<td>2 clock cycles</td>
<td></td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>Unified (instruction and data) per core</td>
<td>Unified (instruction and data) per core</td>
<td></td>
</tr>
<tr>
<td>L2 block size</td>
<td>256 KB (0.2 MB)</td>
<td>512 KB (0.5 MB)</td>
<td></td>
</tr>
<tr>
<td>L2 cache associativity</td>
<td>8-way set associative</td>
<td>8-way set associative</td>
<td></td>
</tr>
<tr>
<td>L2 replacement</td>
<td>Approximated LRU replacement</td>
<td>Approximated LRU replacement</td>
<td></td>
</tr>
<tr>
<td>L2 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
<td></td>
</tr>
<tr>
<td>L2 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
<td></td>
</tr>
<tr>
<td>L2 hit time</td>
<td>Not Available</td>
<td>8 clock cycles</td>
<td></td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>Unified (instruction and data)</td>
<td>Unified (instruction and data)</td>
<td></td>
</tr>
<tr>
<td>L3 cache size</td>
<td>8192 KB (8 MB), shared</td>
<td>2048 KB (2 MB), shared</td>
<td></td>
</tr>
<tr>
<td>L3 cache associativity</td>
<td>16-way set associative</td>
<td>32-way set associative</td>
<td></td>
</tr>
<tr>
<td>L3 replacement</td>
<td>Not Available</td>
<td>Evict block shared by fewest cores</td>
<td></td>
</tr>
<tr>
<td>L3 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
<td></td>
</tr>
<tr>
<td>L3 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
<td></td>
</tr>
<tr>
<td>L3 hit time</td>
<td>Not Available</td>
<td>3B (75) clock cycles</td>
<td></td>
</tr>
</tbody>
</table>

• 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