Review: Performance

- Latency vs. Throughput.
- Time (seconds/program) is performance measure
  \[ \text{Instructions} \times \frac{\text{Seconds}}{\text{Program}} \]
  \[ \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Clock Cycle}}{\text{Second}} \]
- Time measurement via clock cycles, machine specific
- Power of increasing concern, and being added to benchmarks
- Profiling tools (e.g., gprof) as way to see where spending time in your program.

Agenda

- Memory Hierarchy Overview and Analogy
- Administrivia
- Direct Mapped Caches
- Break
- Direct Mapped Cache Example
- Cache Performance

Storage in a Computer

- Processor
  - Holds data in register file (~100 Bytes)
  - Registers accessed on sub-nanosecond timescale
- Memory (we’ll call “main memory”)
  - More capacity than registers (~Gbytes)
  - Access time ~50-100 ns
  - Hundreds of clock cycles per memory access?!
Great Idea #3: Principle of Locality/Memory Hierarchy

Library Analogy

- Writing a report on a specific topic.
  - E.g., works of J.D. Salinger
- While at library, check out books and keep them on desk.
  - If need more, check them out and bring to desk.
    - But don’t return earlier books since might need them
    - Limited space on desk; Which books to keep?
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

Locality

- **Temporal Locality** (locality in time)
  - Go back to same book on desktop multiple times
  - If a memory location is referenced then it will tend to be referenced again soon
- **Spatial Locality** (locality in space)
  - When go to book shelf, pick up multiple books on J.D. Salinger since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Principle of Locality

- **Principle of Locality**: Programs access small portion of address space at any instant of time
- What program structures lead to temporal and spatial locality in code?
- In data?

How does hardware exploit principle of locality?

- Offer a hierarchy of memories where
  - closest to processor is fastest (and most expensive per bit so smallest)
  - furthest from processor is largest (and least expensive per bit so slowest)
- Goal is to create illusion of memory almost as fast as fastest memory and almost as large as biggest memory of the hierarchy

Memory Hierarchy

As we move to deeper levels the latency goes up and price per bit goes down. Why?
Caches

- Processor and memory speed mismatch leads us to add a new level: a memory cache
- Implemented with same integrated circuit processing technology as processor, integrated on-chip: faster but more expensive than DRAM memory
- **Cache is a copy of a subset of main memory**
- Modern processors have separate caches for instructions and data, as well as several levels of caches implemented in different sizes
- As a pun, often use $ ("cash") to abbreviate cache, e.g. D$ = Data Cache, I$ = Instruction Cache

Memory Hierarchy Technologies

- Caches use SRAM (Static RAM) for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 ns)
  - Low density (6 transistor cells), higher power, expensive ($2000 to $4000 per GB in 2011)
  - Static: content will last as long as power is on
- Main memory uses DRAM (Dynamic RAM) for size (density)
  - Slower (typical access times of 50 to 70 ns)
  - High density (1 transistor cells), lower power, cheaper ($20 to $40 per GB in 2011)
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
- Consumes 1% to 2% of the active cycles of the DRAM

Characteristics of the Memory Hierarchy

- **Block** - Unit of transfer between memory and cache
- Increasing distance from the processor in access time
- Inclusive—what is in L1S is a subset of what is in L2S is a subset of what is in MM that is a subset of is in SM
- (Relative) size of the memory at each level

How is the Hierarchy Managed?

- **registers ↔ memory**
  - By compiler (or assembly level programmer)
- **cache ↔ main memory**
  - By the cache controller hardware
- **main memory ↔ disks (secondary storage)**
  - By the operating system (virtual memory)
  - (Talk about later in the semester)
  - Virtual to physical address mapping assisted by the hardware (TLB)
  - By the programmer (files)

Typical Memory Hierarchy

- On-Chip Components: unit of transfer between memory and cache
- Speed (cycles): 3% 1% 10% 100% 1000,000%
- Size (bytes): 100% 100K's M's G's T's
- Cost/bit: highest lowest
- Principle of locality + memory hierarchy presents programmer with as much memory as is available in the cheapest technology at the speed offered by the fastest technology

Review so far

- Wanted: size of the largest memory available, speed of the fastest memory available
- Approach: Memory Hierarchy
  - Successively lower levels contain “most used” data from next higher level
  - Exploits temporal & spatial locality
Agenda

- Memory Hierarchy Overview and Analogy
- Administrivia
- Direct Mapped Caches
- Break
- Direct Mapped Cache Example
- Cache Performance

Administrivia

- Midterm
  - Friday 7/15, 9am-12pm, 2050 VLSB
  - How to study:
    - Studying in groups can help.
    - Take old exams for practice (link at top of main webpage)
    - Look at lectures, section notes, projects, hw, labs, etc.
    - Go to Review Session.
  - Will cover up to tomorrow's material.
- Midterm Review Session
  - TODAY, 4pm – 6pm, Wozniak Lounge

Cache Management

- Cache managed automatically by hardware.
- Operations available in hardware are limited, scheme needs to be relatively simple.
- Where in the cache do we put a block of data from memory?
  - How do we find it when we need it?
- What is the overall organization of blocks we impose on our cache?

Direct Mapped Caches

- Each memory block is mapped to exactly one block in the cache
  - Only need to check this single location to see if block is in cache.
- Cache is smaller than memory
  - Multiple blocks in memory map to a single block in the cache!
  - Need some way of determining the identity of the block.
Direct Mapped Caches

- Address mapping:
  - (block address) modulo (# of blocks in the cache)
  - Lower bits of memory address determine which block in the cache the block is stored.
  - Upper bits of memory address (Tag) determine which block in memory the block came from.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Address Fields (For now...)</td>
<td></td>
</tr>
</tbody>
</table>

7/14/2011
Spring 2011 -- Lecture #11

Full Address Breakdown

- Lowest bits of address (Offset) determine which byte within a block it refers to.
- Full address format:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- n-bit Offset means a block is how many bytes?
- n-bit Index means cache has how many blocks?

7/14/2011
Spring 2011 -- Lecture #11

Caching: A First Example

Main Memory - 6 bit addresses

<table>
<thead>
<tr>
<th>Cache</th>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td></td>
<td>0000x</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td></td>
<td>0011x</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>0111x</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>1111x</td>
</tr>
</tbody>
</table>

Q: Where in the cache is the mem block?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

7/14/2011
Spring 2011 -- Lecture #11

Block Mapping From Memory

<table>
<thead>
<tr>
<th>Cache</th>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>1011</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>1111</td>
</tr>
</tbody>
</table>

4-bit memory addresses

4-bit memory addresses

<table>
<thead>
<tr>
<th>Cache</th>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>1011</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>1111</td>
</tr>
</tbody>
</table>

(block address) modulo (# of blocks in the cache)

7/14/2011
Spring 2011 -- Lecture #11

TIO Breakdown - Summary

- All fields are read as unsigned integers.
  - Index
    - specifies the cache index (which “row”/block of the cache we should look in)
    - 1 bits => 2^I blocks in cache
  - Offset
    - once we’ve found correct block, specifies which byte within the block we want (which “column” in the cache)
    - 0 bits => 2^O bytes per block
  - Tag
    - the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to a given location

7/14/2011
Spring 2011 -- Lecture #11

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words

<table>
<thead>
<tr>
<th>Cache</th>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td></td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>1110</td>
</tr>
</tbody>
</table>

7/14/2011
Spring 2011 -- Lecture #11
Caching Terminology

• When reading memory, 3 things can happen:
  – cache hit: cache block is valid and contains proper address, so read desired word
  – cache miss: nothing in cache in appropriate block, so fetch from memory
  – cache miss, block replacement: wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

Agenda

• Memory Hierarchy Overview and Analogy
• Administrivia
• Direct Mapped Caches
• Break
• Direct Mapped Cache Example
• Cache Performance

Direct Mapped Cache

- Consider the sequence of memory address accesses
  Start with an empty cache - all blocks initially marked as not valid

<table>
<thead>
<tr>
<th>Time</th>
<th>0 miss</th>
<th>1 miss</th>
<th>2 miss</th>
<th>3 miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
<td>Mem(0)</td>
</tr>
<tr>
<td>01</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
<td>Mem(1)</td>
</tr>
<tr>
<td>02</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
<td>Mem(2)</td>
</tr>
<tr>
<td>03</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
<td>Mem(3)</td>
</tr>
<tr>
<td>04</td>
<td>Mem(4)</td>
<td>Mem(4)</td>
<td>Mem(4)</td>
<td>Mem(4)</td>
</tr>
<tr>
<td>05</td>
<td>Mem(5)</td>
<td>Mem(5)</td>
<td>Mem(5)</td>
<td>Mem(5)</td>
</tr>
<tr>
<td>06</td>
<td>Mem(6)</td>
<td>Mem(6)</td>
<td>Mem(6)</td>
<td>Mem(6)</td>
</tr>
<tr>
<td>07</td>
<td>Mem(7)</td>
<td>Mem(7)</td>
<td>Mem(7)</td>
<td>Mem(7)</td>
</tr>
</tbody>
</table>

Tate: 8 requests, 6 misses

Miss Rate vs Block Size vs Cache Size

- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)
Agenda
• Memory Hierarchy Overview and Analogy
• Administrivia
• Direct Mapped Caches
• Break
• Direct Mapped Cache Example
• Cache Performance

Average Memory Access Time (AMAT)
• Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses
  \[ \text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty} \]
• What is the AMAT for a processor with a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?
  \[ 1 + 0.02 \times 50 = 2 \text{ clock cycles} \]
  Or \[ 2 \times 200 = 400 \text{ psecs} \]
• Potential impact of much larger cache on AMAT?
  1) Lower Miss rate
  2) Longer Access time (Hit time): smaller is faster
     Increase in hit time will likely add another stage to the pipeline
     At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance

Measuring Cache Performance – Effect on CPI
• Assuming cache hit costs are included as part of the normal CPU execution cycle, then
  \[ \text{CPU time} = IC \times \text{CPI} \times \text{CC} \]
  \[ = IC \times (\text{CPI}_{\text{ideal}} + \text{Average Memory-stall cycles}) \times \text{CC} \]
• A simple model for Memory-stall cycles
  Memory-stall cycles = accesses/instruction \times \text{miss rate} \times \text{miss penalty}
  • Will talk about writes and write misses next lecture, where its a little more complicated

Impacts of Cache Performance
• Relative $ penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  – Memory speed unlikely to improve as fast as processor cycle time. When calculating CPI_{ideal} cache miss penalty is measured in processor clock cycles needed to handle a miss
  – Lower the CPI_{ideal}, more pronounced impact of stalls
• Processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr’s, and 2% I$ and 4% D$ miss rates
  – Memory-stall cycles = 2% \times 100 + 36% \times 4% \times 100 = 3.44
  – So \[ \text{CPI}_{\text{ideal}} = \frac{2 + 3.44}{5.44} \]
  – More than twice the CPI_{ideal}!
• What if the CPI_{ideal} is reduced to 1?
• What if the D$ miss rate went up by 1%?

“And In Conclusion..”
• Principle of Locality
• Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
• Direct Mapped Cache – Each block in memory maps to one block in the cache.
  – Index to determine which block.
  – Offset to determine which byte within block
  – Tag to determine if it’s the right block.
• AMAT to measure cache performance