CPE 631 Lecture 04: CPU Caches

Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

- Memory Hierarchy
- Four Questions for Memory Hierarchy
- Cache Performance
Processor-DRAM Latency Gap

- Processor: 2x/1.5 year
- Memory: 2x/10 years
- Processor-Memory Performance Gap grows 50% / year

Solution: The Memory Hierarchy (MH)

User sees as much memory as is available in cheapest technology and access it at the speed offered by the fastest technology.

Levels in Memory Hierarchy:
- Upper
- Lower

Processor
Control
Datapath

Speed: Fastest to Slowest
Capacity: Smallest to Biggest
Cost/bit: Highest to Lowest
Generations of Microprocessors

Time of a full cache miss in instructions executed:
1st Alpha: 340 ns/5.0 ns = 68 clks x 2 or 136
2nd Alpha: 266 ns/3.3 ns = 80 clks x 4 or 320
3rd Alpha: 180 ns/1.7 ns = 108 clks x 6 or 648

1/2X latency x 3X clock rate x 3X Instr/clock ⇒ -5X

Why hierarchy works?

- Principal of locality

- Temporal locality: recently accessed items are likely to be accessed in the near future ⇒ Keep them close to the processor

- Spatial locality: items whose addresses are near one another tend to be referenced close together in time ⇒ Move blocks consisted of contiguous words to the upper level

Rule of thumb: Programs spend 90% of their execution time in only 10% of code
Cache Measures

- **Hit**: data appears in some block in the upper level (Bl. X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: time to access the upper level (RAM access time + Time to determine hit/miss)
- **Miss**: data needs to be retrieved from the lower level (Bl. Y)
  - Miss rate: 1 - (Hit Rate)
  - Miss penalty: time to replace a block in the upper level + time to retrieve the block from the lower level
- **Average memory-access time**
  \[ \text{Average memory-access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty} \] (ns or clocks)

Hit time << Miss Penalty

Levels of the Memory Hierarchy

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Access Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Registers</td>
<td>&lt;1s ns</td>
<td>$10/ MByte</td>
</tr>
<tr>
<td>Cache</td>
<td>1-10 ns</td>
<td>$10/ MByte</td>
</tr>
<tr>
<td>Main Memory</td>
<td>100ns-300ns</td>
<td>$1/ MByte</td>
</tr>
<tr>
<td>Disk</td>
<td>10s-60s</td>
<td>$0.0031/ MByte</td>
</tr>
<tr>
<td>Tape</td>
<td>infinite</td>
<td>$0.0014/ MByte</td>
</tr>
</tbody>
</table>

Upper Level Memory

- Registers
  - Instr. Operands
- Cache
  - Blocks
- Memory
  - Pages

Lower Level Memory

- Disk
  - Files
- Tape
  - Staging Xfer Unit
  - Upper Level
  - Staging Xfer Unit
  - Upper Level
  - Staging Xfer Unit
  - Upper Level
Four Questions for Memory Heir.

- Q#1: Where can a block be placed in the upper level?
  ⇒ Block placement
  - direct-mapped, fully associative, set-associative
- Q#2: How is a block found if it is in the upper level?
  ⇒ Block identification
- Q#3: Which block should be replaced on a miss?
  ⇒ Block replacement
  - Random, LRU (Least Recently Used)
- Q#4: What happens on a write?
  ⇒ Write strategy
  - Write-through vs. write-back
  - Write allocate vs. No-write allocate

Direct-Mapped Cache

- In a direct-mapped cache, each memory address is associated with one possible block within the cache
  - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  - Block is the unit of transfer between cache and memory
Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

<table>
<thead>
<tr>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Full Mapped
  - (12 mod 8) = 4
- Direct Mapped
  - (12 mod 8) = 4
- 2-Way Assoc
  - (12 mod 4) = 0

---

Direct-Mapped Cache (cont’d)

Memory Address

0 1 2 3 4 5 6 7 8 9 A B C D E F

Memory

- Cache Index
  - 0 1 2 3

Cache (4 byte)
Direct-Mapped Cache (cont’d)

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- What if we have a block size > 1 byte?
- Result: divide memory address into three fields:

  Block Address

  tttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
Direct-Mapped Cache Example

- Conditions
  - 32-bit architecture (word=32 bits), address unit is byte
  - 8KB direct-mapped cache with 4 words blocks

- Determine the size of the Tag, Index, and Offset fields
  - OFFSET (specifies correct byte within block): cache block contains 4 words = 16 (2^4) bytes ⇒ 4 bits
  - INDEX (specifies correct row in the cache): cache size is 8KB=2^{13} bytes, cache block is 2^4 bytes
    #Rows in cache (1 block = 1 row): 2^{13}/2^4 = 2^9 ⇒ 9 bits
  - TAG: Memory address length - offset - index =
    32 - 4 - 9 = 19 ⇒ tag is leftmost 19 bits

1 KB Direct Mapped Cache, 32B blocks

- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operate in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result

Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.
Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

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

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Assoc:</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>LRU Ran 4-way</td>
<td>LRU Ran 4-way</td>
<td>LRU Ran 8-way</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2% 5.7%</td>
<td>4.7% 5.3%</td>
<td>4.4% 5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9% 2.0%</td>
<td>1.5% 1.7%</td>
<td>1.4% 1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15% 1.17%</td>
<td>1.13% 1.13%</td>
<td>1.12% 1.12%</td>
</tr>
</tbody>
</table>
Q4: What happens on a write?

- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.
- **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- **Pros and Cons of each?**
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- **WT always combined with write buffers so that don’t wait for lower level memory**

Write stall in write through caches

- When the CPU must wait for writes to complete during write through, the CPU is said to write stall
- **Common optimization**
  => Write buffer which allows the processor to continue as soon as the data is written to the buffer, thereby overlapping processor execution with memory updating
- However, write stalls can occur even with write buffer (when buffer is full)
Write Buffer for Write Through

- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) \ll \frac{1}{\text{DRAM write cycle}}
- Memory system designer’s nightmare:
  - Store frequency (w.r.t. time) \rightarrow \frac{1}{\text{DRAM write cycle}}
  - Write buffer saturation

What to do on a write-miss?

- Write allocate (or fetch on write)
  The block is loaded on a write-miss, followed by the write-hit actions
- No-write allocate (or write around)
  The block is modified in the memory and not loaded into the cache
- Although either write-miss policy can be used with write through or write back, write back caches generally use write allocate and write through often use no-write allocate
An Example: The Alpha 21264 Data Cache (64KB, 64-byte blocks, 2w)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Tag</th>
<th>Index</th>
<th>CPUAddress</th>
<th>Data in</th>
<th>Data out</th>
<th>Write buffer</th>
<th>Lower level memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;29&gt;</td>
<td>&lt;9&gt;</td>
<td>&lt;6&gt;</td>
<td></td>
<td>Valid&lt;1&gt; Tag&lt;29&gt; Data&lt;512&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td>8:1 Mux</td>
<td>2:1 MUX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
<td>8:1 Mux</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Performance

- **Hit Time** = time to find and retrieve data from current level cache
- **Miss Penalty** = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- **Hit Rate** = % of requests that are found in current level cache
- **Miss Rate** = 1 - Hit Rate
Cache Performance (cont’d)

- Average memory access time (AMAT)

\[
AMAT = Hit\ time + Miss\ Rate \times Miss\ Penalty
\]

\[
= \%\ instructions \times (Hit\ time_{inst} + Miss\ Rate_{inst} \times Miss\ Penalty_{inst})
\]

\[
+ \%\ data \times (Hit\ time_{data} + Miss\ Rate_{data} \times Miss\ Penalty_{data})
\]

An Example: Unified vs. Separate I&D

- Compare 2 design alternatives (ignore L2 caches)?
  - 16KB I&D: Inst misses=3.82/1K, Data miss rate=40.9/1K
  - 32KB unified: Unified misses = 43.3 misses/1K

- Assumptions:
  - ld/st frequency is 36% → 74% accesses from instructions (1.0/1.36)
  - hit time = 1 clock cycle, miss penalty = 100 clock cycles
  - Data hit has 1 stall for unified cache (only one port)
Unified vs. Separate I&D (cont’d)

- Miss rate (L1I) = (# L1I misses) / (IC)
- #L1I misses = (L1I misses per 1k) * (IC / 1000)
- Miss rate (L1I) = 3.82/1000 = 0.0038

- Miss rate (L1D) = (# L1D misses) / (# Mem. Refs)
- #L1D misses = (L1D misses per 1k) * (IC / 1000)
- Miss rate (L1D) = 40.9 * (IC/1000) / (0.36*IC) = 0.1136

- Miss rate (L1U) = (# L1U misses) / (IC + Mem. Refs)
- #L1U misses = (L1U misses per 1k) * (IC / 1000)
- Miss rate (L1U) = 43.3*(IC / 1000) / (1.36 * IC) = 0.0318

 Unified vs. Separate I&D (cont’d)

- AMAT (split) = (% instr.) * (hit time + L1I miss rate * Miss Pen.) + (% data) * (hit time + L1D miss rate * Miss Pen.) = .74(1 + .0038*100) + .26(1+.1136*100) = 4.2348 clock cycles

- AMAT (unif.) = (% instr.) * (hit time + L1Umiss rate * Miss Pen.) + (% data) * (hit time + L1U miss rate * Miss Pen.) = .74(1 + .0318*100) + .26(1 + 1 + .0318*100) = 4.44 clock cycles
AMAT and Processor Performance

- Miss-oriented Approach to Memory Access
  - $\text{CPI}_{\text{Exec}}$ includes ALU and Memory instructions

\[
\text{CPU time} = \frac{IC \times \left( \frac{\text{CPI}_{\text{Exec}}}{\text{Inst}} + \frac{\text{MemAccess}}{\text{Inst}} \times \text{MissRate} \times \text{MissPenalty} \right)}{\text{Clock rate}}
\]

AMAT and Processor Performance (cont’d)

- Separating out Memory component entirely
  - $\text{AMAT} = \text{Average Memory Access Time}$
  - $\text{CPI}_{\text{ALUOps}}$ does not include memory instructions

\[
\text{CPU time} = \frac{IC \times \left( \frac{\text{ALUops}}{\text{Inst}} \times \text{CPI}_{\text{ALUops}} + \frac{\text{MemAccess}}{\text{Inst}} \times \text{AMAT} \right)}{\text{Clock rate}}
\]

\[
\text{AMAT} = \text{Hit time} + \text{Miss Rate} \times \text{Miss Penalty} = \% \text{ instructions} \times (\text{Hit time}_{\text{inst}} + \text{Miss Rate}_{\text{inst}} \times \text{Miss Penalty}_{\text{inst}}) + \% \text{ data} \times (\text{Hit time}_{\text{Data}} + \text{Miss Rate}_{\text{Data}} \times \text{Miss Penalty}_{\text{Data}})
\]
Summary: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space

- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity

- Write Policy:
  - Write Through: needs a write buffer.
  - Write Back: control can be complex

- Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?

Summary: 
The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back

- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins