Recap: Who Cares About the Memory Hierarchy?
Processor-DRAM Memory Gap (latency)

Moore's Law

Processor-Memory Performance Gap:
- Grows 50% / year
- CPU 60%/yr. (2X/1.5 yr)
- DRAM 9%/yr. (2X/10 yrs)
Levels of the Memory Hierarchy

- **Capacity**
  - CPU Registers: 100s Bytes, <10s ns
  - Cache: K Bytes, 10-100 ns, 1-0.1 cents/bit
  - Main Memory: M Bytes, 200ns - 500ns, $0.001 - 0.0001 cents/bit
  - Disk: 6 Bytes, 10 ms, (10,000,000 ns), -5 - 6, 10 - 10 cents/bit
  - Tape: infinite sec, $0.001 - 0.0001 cents/bit

- **Access Time**
  - Faster (100s-500ns)
  - Infinite (100s-500ns)

- **Files**
  - Pages
  - Instr. Operands

The Principle of Locality

- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.

- **Two Different Types of Locality:**
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** ( Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

- **Last 15 years, HW relied on locality for speed**
  - It is a property of programs which is exploited in machine design.
Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)
Simplest Cache: Direct Mapped

- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

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)

<table>
<thead>
<tr>
<th>Valid Bit</th>
<th>Cache Tag</th>
<th>Cache Tag Example: 0x50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stored as part of the cache “state”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x50</td>
<td>0</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 31: ** Byte 1: Byte 0: 0</td>
</tr>
<tr>
<td>Byte 63: ** Byte 33: Byte 32: 1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>Byte 1023: ** Byte 992: 31</td>
</tr>
</tbody>
</table>
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates 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
4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

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>Full Mapped</th>
<th>Direct Mapped (12 mod 8) = 4</th>
<th>2-Way Assoc (12 mod 4) = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01234567</td>
<td>01234567</td>
<td>01234567</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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: 2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>LRU</td>
<td>Ran</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</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 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) \( << 1 / \text{DRAM write cycle} \)
- Memory system designer’s nightmare:
  - Store frequency (w.r.t. time) \( -> 1 / \text{DRAM write cycle} \)
  - Write buffer saturation
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  - What does this mean to Compilers, Data structures, Algorithms?
- Quicksort: fastest comparison based sorting algorithm when all keys fit in memory
- Radix sort: also called “linear time” sort because for keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
- For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

A Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.
Virtual memory => treat memory as a cache for the disk
• Terminology: blocks in this cache are called “Pages”
  - Typical size of a page: 1K – 8K
• Page table maps virtual page numbers to physical frames
  - “PTE” = Page Table Entry

Three Advantages of Virtual Memory

• Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program (“Working Set”) must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.

• Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    • (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs

• Sharing:
  - Can map same physical page to multiple users (“Shared memory”)
Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? ⇒ page size
(Contrast with physical block size on disk, i.e. sector size)
Which region of M is to hold the new block ⇒ placement policy
How do we find a page when we look for it? ⇒ block identification
Block of information brought into M, and M is full, then some region of M must be released to make room for the new block ⇒ replacement policy

What do we do on a write? ⇒ write policy

Missing item fetched from secondary memory only on the occurrence of a fault ⇒ demand load policy

Translation Look-Aaside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped
TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.
Cache Performance

Avg. memory access time = Hit time + Miss rate x Miss penalty

with
- Hit time: time to access a data item which is available in the cache
- Miss rate: ratio of no. of memory access leading to a cache miss to the total number of instructions
- Miss penalty: time/cycles required for making a data item in the cache

Split vs. unified cache

- Assume two machines:
  - Machine 1: 16KB instruction cache + 16 KB data cache
  - Machine 2: 32KB unified cache
- Assume for both machines:
  - 36% of instructions are memory references/data transfers
  - 74% of memory references are instruction references
  - Misses per 1000 instructions:
    - 16 KB instruction cache: 3.82
    - 16 KB data cache: 40.9
    - 32 KB unified cache: 43.3
  - Hit time:
    - 1 clock cycle for machine 1
    - 1 additional clock cycle for machine 2 for data accesses (structural hazard)
  - Miss penalty: 100 clock cycles
Split vs. unified cache (II)

- Questions:
  1. Which architecture has a lower miss-rate?
  2. What is the average memory access time for both machines?

Miss-rate per instruction can be calculated as:

$$\text{Miss rate} = \frac{(\text{Misses})}{1000}$$

$$\text{Memory access}$$

$$\text{Instruction}$$

Split vs. unified cache (III)

- Machine 1:
  - since every instruction access requires exactly one memory access:
    $$\text{Miss rate}_{16 \text{ KB instruction}} = \frac{(3.82/1000)}{1.0} = 0.00382 \approx 0.004$$
  
  - Since 36% of the instructions are data transfer:
    $$\text{Miss rate}_{16 \text{ KB data}} = \frac{(40.9/1000)}{0.36} = 0.114$$
  
  - Overall miss rate: since 74% of memory access are instructions references:
    $$\text{Miss rate}_{\text{split cache}} = (0.74 * 0.004) + (0.26 * 0.114) = 0.0324$$
Split vs. unified cache (IV)

- Machine 2:
  - Unified cache needs to account for the instruction fetch and data access
    \[
    \text{Miss rate}_{32KB\text{ unified}} = \frac{43.4}{1000} / (1 + 0.36) = 0.0318
    \]
  → Answer to question 1: the 2\textsuperscript{nd} architecture has a lower miss rate

Split vs. unified cache (V)

- Average memory access time (AMAT):
  \[
  \text{AMAT} = \%\text{instructions} \times (\text{Hit time} + \text{Instruction Miss rate} \times \text{Miss penalty}) +
  \%\text{data} \times (\text{Hit time} + \text{Data Miss rate} \times \text{Miss penalty})
  \]
  - Machine 1:
    \[
    AMAT_1 = 0.74 \times (1 + 0.004 \times 100) + 0.26 \times (1 + 0.114 \times 100) = 4.24
    \]
  - Machine 2:
    \[
    AMAT_2 = 0.74 \times (1 + 0.0318 \times 100) + 0.26 \times (1 + 1 + 0.0318 \times 100) = 4.44
    \]
  → Answer to question 2: the 1st machine has a lower average memory access time
Direct mapped vs. set associative

- **Assumptions:**
  - CPI without cache misses ( = perfect cache) : 2.0
  - No. of memory references per instruction: 1.5
  - Cache size: 64 KB
    - **Machine 1:** direct mapped cache
      - Clock cycle time: 1 ns
      - Miss rate: 1.4%
    - **Machine 2:** 2-way set associative
      - Clock cycle time: 1.25 ns
      - Miss rate: 1.0%
  - Cache miss penalty: 75 ns
  - Hit time: 1 clock cycle

- **AMAT:** Hit time + (Miss rate x Miss penalty)

\[
\text{AMAT}_1 = 1.0 + (0.014 \times 75) = 2.05 \text{ ns} \\
\text{AMAT}_2 = 1.25 + (0.010 \times 75) = 2.0 \text{ ns}
\]

→ avg. memory access time better for 2-way set associative cache
Direct mapped vs. set associative (III)

- CPU performance:
  
  \[
  \text{CPU time} = IC \times (\text{CPI}_{\text{exec}} + \text{Misses/instruction} \times \text{Miss penalty}) \times \text{Clock cycle time}
  \]

  \[
  = IC \times [(\text{CPI}_{\text{exec}} \times \text{Clock cycle time}) + (\text{Miss rate} \times \text{memory access/instruction} \times \text{Miss penalty} \times \text{Clock cycle time})]
  \]

  CPU time 1 = IC \times (2 \times 1.0 + (1.5 \times 0.014 \times 75 \times 1.0)) = 3.58 IC
  CPU time 2 = IC \times (2 \times 1.25 + (1.5 \times 0.01 \times 75 \times 1.25)) = 3.63 IC

  \[\rightarrow\] Direct mapped cache leads to better CPU time