Recap: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)

“Moore’s Law”

Processor performance:
- μProc: 60%/yr. (2X/1.5yr)

DRAM performance:
- 9%/yr. (2X/10 yrs)

Processor-Memory Performance Gap:
- grows 50% / year
Levels of the Memory Hierarchy

Capacity
Access Time
Cost

**CPU Registers**
100s Bytes
<10s ns

**Cache**
K Bytes
10-100 ns
1-0.1 cents/bit

**Main Memory**
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit

**Disk**
G Bytes, 10 ms
(10,000,000 ns)
$10^{-5}$ - 10 cents/bit

**Tape**
infinite
sec-min
10^{-8}

---

**Upper Level**

**Registers**

**Instr. Operands**

**Cache**

**Blocks**

**Memory**

**Pages**

**Disk**

**Files**

**Tape**

**Staging Xfer Unit**

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes

---

**Lower Level**

faster

Larger
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!)
Cache Measures

- **Hit rate**: fraction found in that level
  - So high that usually talk about **Miss rate**
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory

- **Average memory-access time**
  \[
  = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}
  \]
  (ns or clocks)

- **Miss penalty**: time to replace a block from lower level, including time to replace in CPU
  - **access time**: time to lower level
    \[
    = f(\text{latency to lower level})
    \]
  - **transfer time**: time to transfer block
    \[
    = f(\text{BW between upper & lower levels})
    \]
Simplest Cache: Direct Mapped

Memory Address | Memory
---|---
0 | 4 Byte Direct Mapped Cache
1 |
2 |
3 |
4 |
5 |
6 |
7 |
8 |
9 |
A |
B |
C |
D |
E |
F |

4 Byte Direct Mapped Cache

- 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 \(2^{\text{**}N}\) byte cache:
  – The uppermost \((32 - N)\) bits are always the Cache Tag
  – The lowest \(M\) bits are the Byte Select (Block Size = \(2^{\text{**}M}\))

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Example: 0x50</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ex: 0x01</td>
<td>Ex: 0x00</td>
</tr>
</tbody>
</table>

**Stored as part of the cache “state”**

- Valid Bit
- Cache Tag
- Cache Data

- Byte 31
- Byte 63
- Byte 1023

- Byte 0
- Byte 1
- Byte 32
- Byte 992

- Byte 31
- Byte 63
- Byte 1023

- Byte 0
- Byte 1
- Byte 32
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
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if 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>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>01234567</td>
<td>01234567</td>
<td>01234567</td>
</tr>
</tbody>
</table>

Memory

COSC 6385 – Computer Architecture
Edgar Gabriel
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</td>
<td>Ran</td>
<td>LRU</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</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 / DRAM write cycle
- Memory system designer’s nightmare:
  - Store frequency (w.r.t. time) -> 1 / 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
Quicksort vs. Radix as vary number keys: Instructions

![Graph showing comparison between Quicksort and Radix sort]
Quicksort vs. Radix as vary number keys: Instrs & Time

![Graph comparing Quicksort and Radix sort with set size in keys, instructions, and time. Red line for Quick sort, blue for Radix sort, magenta for Quick time, and cyan for Radix time. The x-axis represents set size in keys, ranging from 1000 to 1E+07, and the y-axis represents instructions and time.]
Quicksort vs. Radix as vary number keys: Cache misses

What is proper approach to fast algorithms?
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.

<table>
<thead>
<tr>
<th>Speed (ns)</th>
<th>1s</th>
<th>10s</th>
<th>100s</th>
<th>10,000,000s</th>
<th>10,000,000,000s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (bytes):</td>
<td>100s</td>
<td>Ks</td>
<td>Ms</td>
<td>Gs</td>
<td></td>
</tr>
</tbody>
</table>

Control
Datapath
Registers
On-Chip Cache
Second Level Cache (SRAM)
Main Memory (DRAM)
Secondary Storage (Disk)
Tertiary Storage (Disk/Tape)
What is virtual memory?

- 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
    => Far more “viruses” under Microsoft Windows

• **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-Aside 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.
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.
    Nightmare Scenario: ping pong effect!

• Write Policy:
  – **Write Through**: needs a **write buffer**. Nightmare: WB saturation
  – **Write Back**: control can be complex
Summary: The Cache Design Space

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

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

- Simplicity often wins
Summary: TLB, Virtual Memory

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions: 1) Where can block be placed? 2) How is block found? 3) What block is replaced on miss? 4) How are writes handled?
- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can’t access all of 2nd level cache without TLB misses!
Summary : Memory Hierarchy

• Virtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
  – 1000X DRAM growth removed the controversy
• Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy
• Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  What does this mean to Compilers, Data structures, Algorithms?