Hardware and The Memory Hierarchy

Andrew Case

Slides adapted from Jinjang Li, Mohamed Zahran, Randy Bryant and Dave O’Hallaron
Topics

- **System Hardware**
  - Chipsets
  - Buses
  - I/O devices
  - Memory

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy
System Board (i.e. motherboard)

- External I/O Ports (mouse, network, etc.)
- Memory slots
- Power connector
- Internal I/O Ports (Flash/Disk drives)
- 1 PCIe expansion slot (graphics, etc.)
- CPU socket
- Chipset provides bus interface between devices including CPU, sound, network, graphics, etc.
Chipset

- Provides interface between CPU and other devices
- Designed for use with a specific family of processors.
  - **Northbridge**: connects the CPU to high-speed components like RAM.
  - **Southbridge**: connects to slower peripheral devices like PCI slots, USB, SATA, ...
- Nowadays, northbridge is usually integrated with the CPU chip.
Example chipset: Intel H87

Northbridge integrated with CPU chip

Graphics connection if graphic is via PCIe

20Gbps link

Southbridge functionalities
Buses

• Connect components together
  ▪ Most buses use common protocols for interoperability
• A bus carries address, data, and control lines.
• Common buses:
  ▪ Memory bus
  ▪ PCI bus
  ▪ SATA
  ▪ Universal serial bus (USB)
  ▪ System management bus (SM)
Traditional Bus Structure Connecting CPU and Memory
I/O Bus

Expansion slots for other devices such as network adapters.
Buses

- Most buses are synchronous (both ends are synchronized)
- Some require external clock signal
  - A separate line carries clock signal
  - All activities take an integral number of clock cycles
  - Examples:
    - Intel’s memory bus operates at 2,4, 3.2, or 4GHz
- Some require no external clock signal
  - Data encapsulated in a “packet”
    - Packets start with preamble for the receiver to synchronize
  - Examples:
    - USB
    - PCI
    - SATA
Bus Example: PCI-e

- A high-performance bus for peripherals.
  - Graphics cards
  - Network cards

- A serial point-to-point interconnect between two devices
  - Serial - one bit at a time (as opposed to in parallel)
  - No shared bus but a shared switch

- Data sent in packets
  - Synchronization information embedded in signal
Bandwidth of PCI-e

<table>
<thead>
<tr>
<th>Link Width</th>
<th>x1</th>
<th>x2</th>
<th>x4</th>
<th>x8</th>
<th>x12</th>
<th>x16</th>
<th>x32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate BW (GBytes/s)</td>
<td>0.5</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>
PCIe Card

- Is any device connected to PCIe bus

Graphics card PCIe (x16)
Bus Example: USB

- Universal Serial Bus
- Communicates both data and power
- USB 3.0 has a transfer rate up to 5Gbps
Bus Example: USB

- **Host**: Initiates transactions and bandwidth usage.
  - Most complexity lies at host controllers → allowing for cheaper slave devices

- **Slave**: A peripheral USB device
  - Web cam, keyboard, external disks, phone

- **Hub**: A device that contains multiple ports.
  - Root hub connects to the host.
  - We can have a tree of hubs till 5 levels.
Bus Example: USB

- Host
- Root Hub
- HUB
  - Keyboard
  - Disk
  - Speaker
  - Printer
Disk Storage

- 2 Common types of “disk” storage
  - Hard Disk Drive (HDD)
    - Magnetic disks with physical
  - Solid State Disk (SSD)
    - Really is just Flash memory
What’s Inside A Disk Drive?

- Spindle
- Arm
- Actuator
- Platters
- I/O connector
- Electronics (including a processor and memory!)

Image courtesy of Seagate Technology
Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**.
- Each track consists of **sectors** separated by **gaps**.
Disk Capacity

- **Capacity**: maximum number of bits that can be stored (dependent on bit density).
  - Measured in units of gigabytes (GB), where 1 GB = 10^9 Bytes as opposed to 1 GiB = 2^30.
  - \( \text{Capacity} = (\# \text{ bytes/sector}) \times (\text{avg. } \# \text{ sectors/track}) \times (\# \text{ tracks/surface}) \times (\# \text{ surfaces/platter}) \times (\# \text{ platters/disk}) \)

Example:
- 512 bytes/sector x
- 300 sectors/track (on average) x
- 20,000 tracks/surface x
- 2 surfaces/platter x
- 5 platters/disk

\[
\text{Capacity} = 512 \times 300 \times 20000 \times 2 \times 5 = 30,720,000,000 \text{ Bytes}
\]

= 30.72 GB  \( \sim = 28.6 \text{GiB} \)
The disk surface spins at a fixed rotational rate.

The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.
Disk Operation (Multi-Platter View)

Read/write heads move in unison from cylinder to cylinder.
Disk Structure - top view of single platter

Surface organized into tracks

Tracks divided into sectors
Disk Access

Head in position above a track
Disk Access – Read

About to read blue sector
Disk Access – Read

After BLUE read

After reading blue sector
Disk Access – Read

After BLUE read

Red request scheduled next
Disk Access – Seek

After BLUE read

Seek for RED

Seek to red’s track
Disk Access – Rotational Latency

After BLUE read
Seek for RED
Rotational latency

Wait for red sector to rotate around
Disk Access – Read

After BLUE read
Seek for RED
Rotational latency
After RED read

Complete read of red
Disk Access – Service Time Components

- After BLUE read:
  - Data transfer
- Seek for RED:
  - Seek
- Rotational latency:
  - Rotational latency
- After RED read:
  - Data transfer
Disk Access Time

- Based on RPMs
- Throughput: ~50MB/s
- Latency: ~5ms
- Important points:
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.
  - SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower then DRAM.
Solid State Disks (SSDs)

- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after 100,000 repeated writes.
SSD Performance Characteristics

<table>
<thead>
<tr>
<th></th>
<th>Sequential read tput</th>
<th>250 MB/s</th>
<th>Sequential write tput</th>
<th>170 MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random read tput</td>
<td>140 MB/s</td>
<td></td>
<td>Random write tput</td>
<td>14 MB/s</td>
</tr>
<tr>
<td>Rand read access</td>
<td>30 us</td>
<td></td>
<td>Random write access</td>
<td>300 us</td>
</tr>
</tbody>
</table>

- **Why are random writes so slow?**
  - Erasing a block is slow (around 1 ms)
  - Write to a page triggers a copy of all useful pages in the block
    - Find an used block (new block) and erase it
    - Write the page into the new block
    - Copy other pages from old block to the new block
SSD Tradeoffs vs. HDD (Hard Disk Drives)

- **Advantages**
  - No moving parts → faster, less power, more rugged

- **Disadvantages**
  - Have the potential to wear out (so do HDDs)
    - Mitigated by “wear leveling logic” in flash translation layer
    - E.g. Intel X25 guarantees 1 petabyte (1015 bytes) of random writes before they wear out
  - In 2010, about 100x more expensive per byte
  - In 2014, about 3x more expensive per byte

- **Applications**
  - MP3 players, smart phones, laptops
  - Becoming more standard in desktops and servers
I/O Bus

- CPU chip
  - Register file
  - ALU
- System bus
- Memory bus
- Main memory
- I/O bridge
- Bus interface
- I/O bus
- Expansion slots for other devices such as network adapters.

- USB controller
  - Mouse
  - Keyboard
- Graphics adapter
  - Monitor
- Disk controller
  - Disk

- Disk controller
  - Disk
Reading a Disk Sector (1)

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.
When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special “interrupt” pin on the CPU)
Random-Access Memory (RAM)

- **Key features**
  - Basic storage unit is normally a **cell** (one bit per cell).
  - Multiple RAM chips form a memory module.
  - Access any data (random-access) at constant time

- **Static RAM (SRAM)**
  - Each bit stored using multiple-transistor circuit (switch)
  - Retains value indefinitely, as long as it is kept powered.

- **Dynamic RAM (DRAM)**
  - Each cell stores bit with a capacitor. One transistor is used for access
  - Value must be refreshed every 10-100 ms.
## SRAM vs DRAM Summary

<table>
<thead>
<tr>
<th></th>
<th>Trans. per bit</th>
<th>Needs refresh?</th>
<th>Access time</th>
<th>Cost</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SRAM</strong></td>
<td>4 or 6</td>
<td>No</td>
<td>1X</td>
<td>100x</td>
<td>Cache memories</td>
</tr>
<tr>
<td><strong>DRAM</strong></td>
<td>1</td>
<td>Yes</td>
<td>10X</td>
<td>1X</td>
<td>Main memories, etc.</td>
</tr>
</tbody>
</table>
Nonvolatile Memories

- **DRAM and SRAM are volatile memories**
  - Lose information if powered off.

- **Nonvolatile memories retain value even if powered off**
  - Read-only memory (ROM): programmed during production
  - Programmable ROM (PROM): can be programmed once
  - Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
  - Electrically erasable PROM (EEPROM): electronic erase capability
  - **Flash memory**: EEPROMs with partial (sector) erase capability
    - Wears out after about 100,000 erasings.

- **Uses for Nonvolatile Memories**
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)
  - Disk caches
Memory Read Transaction (1)

- CPU places address A on the memory bus.

Load operation: \texttt{movl A, \%eax}
Memory Read Transaction (2)

- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.

Load operation: `movl A, %eax`
Memory Read Transaction (3)

- CPU read word $x$ from the bus and copies it into register `%eax`.
Memory Write Transaction (1)

- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.

Store operation: \texttt{movl \%eax, A}
Memory Write Transaction (2)

- CPU places data word \( y \) on the bus.

Store operation: `movl %eax, A`
Memory Write Transaction (3)

- Main memory reads data word $y$ from the bus and stores it at address $A$.

Store operation: \texttt{movl \%eax, A}
Conventional DRAM Organization

- **D x W DRAM:**
  - bits organized as $D$ supercells of size $w$ bits (size = $D \times W$)
Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.
Step 1(b): Row 2 copied from DRAM array to row buffer.
Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.
Memory Modules

 ADDR (row = i, col = j)

64 MB memory module consisting of eight 8Mx8 DRAMs

Memory controller

64-bit doubleword at main memory address A

64-bit doubleword

: supercell (i,j)
Enhanced DRAMs

- Basic DRAM cell has not changed since its invention in 1966.
- DRAM cores with better interface logic and faster I/O:
  - Synchronous DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR4 (8 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core i7 supports only DDR3 SDRAM
# Storage Trends

## SRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>19,200</td>
<td>2,900</td>
<td>320</td>
<td>256</td>
<td>100</td>
<td>75</td>
<td>60</td>
<td>320</td>
</tr>
<tr>
<td>access (ns)</td>
<td>300</td>
<td>150</td>
<td>35</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>200</td>
</tr>
</tbody>
</table>

## DRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>8,000</td>
<td>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>0.1</td>
<td>0.06</td>
<td>130,000</td>
</tr>
<tr>
<td>access (ns)</td>
<td>375</td>
<td>200</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>9</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>0.064</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2,000</td>
<td>8,000</td>
<td>125,000</td>
</tr>
</tbody>
</table>

## Disk

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>500</td>
<td>100</td>
<td>8</td>
<td>0.30</td>
<td>0.01</td>
<td>0.005</td>
<td>0.0003</td>
<td>1,600,000</td>
</tr>
<tr>
<td>access (ms)</td>
<td>87</td>
<td>75</td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>1</td>
<td>10</td>
<td>160</td>
<td>1,000</td>
<td>20,000</td>
<td>160,000</td>
<td>1,500,000</td>
<td>1,500,000</td>
</tr>
</tbody>
</table>
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.

![Graph showing the CPU-Memory Gap](image-url)

- Disk seek time
- Flash SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time
Locality to the Rescue!

The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality.
Topics

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy
Locality

- **Principle of Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently.

- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future.

- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time.
Locality Example

Data references
- Reference array elements in succession (stride-1 reference pattern).
- Reference variable \( \text{sum} \) each iteration.

Instruction references
- Reference instructions in sequence.
- Cycle through loop repeatedly.

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```
Qualitative Estimates of Locality

- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

- **Question:** Does this function have good locality with respect to array \( a \)?

```c
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    // ij-loop
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```
Question: Does this function have good locality with respect to array \texttt{a}?

```c
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    // ji-loop
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```
Locality Example

**Question:** Can you permute the loops so that the function scans the 3-d array \( a \) with a stride-1 reference pattern (and thus has good spatial locality)?

```c
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];

    return sum;
}
```
Memory Hierarchies

- Some fundamental properties of hardware:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.

- Solution: organize memory and storage systems in a memory hierarchy.
Topics

- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy
An Example Memory Hierarchy

- CPU registers hold words retrieved from L1 cache
- L1 cache holds cache lines retrieved from L2 cache
- L2 cache holds cache lines retrieved from main memory
- Main memory holds disk blocks retrieved from local disks
- Local disks hold files retrieved from disks on remote network servers

Registers

L1 cache (SRAM)

L2 cache (SRAM)

Main memory (DRAM)

Local secondary storage (local disks)

Remote secondary storage (tapes, distributed file systems, Web servers)

Smaller, faster, costlier per byte

Larger, slower, cheaper per byte
# Examples of Caching in the Hierarchy

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>What is Cached?</th>
<th>Where is it Cached?</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-8 bytes words</td>
<td>CPU core</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-bytes block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-bytes block</td>
<td>On/Off-Chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>4-KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware + OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Disk cache</td>
<td>Disk sectors</td>
<td>Disk controller</td>
<td>100,000</td>
<td>Disk firmware</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>
Caches

- **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
Caches

- **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- **Fundamental idea of a memory hierarchy**:
  - For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$. 
Caches

- **Cache:** A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- **Fundamental idea of a memory hierarchy:**
  - For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$.

- **Why do memory hierarchies work?**
  - Because of locality, programs tend to access the data at level $k$ more often than they access the data at level $k+1$.
  - Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit.
Caches

- **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- **Fundamental idea of a memory hierarchy:**
  - For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$.

- **Why do memory hierarchies work?**
  - Because of locality, programs tend to access the data at level $k$ more often than they access the data at level $k+1$.
  - Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit.

- **Big Idea**: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
General Cache Concepts

Cache

Data is copied in block-sized transfer units

Memory

Larger, slower, cheaper memory viewed as partitioned into “blocks”

Smaller, faster, more expensive memory caches a subset of the blocks
General Cache Concepts: Hit

Request: 14

Data in block b is needed

Block b is in cache: Hit!
General Cache Concepts: Miss

Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from memory

Block b is stored in cache
• Placement policy: determines where b goes
• Replacement policy: determines which block gets evicted (victim)
General Caching Concepts:
Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty.

- **Conflict miss**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.
Summary

- The speed gap between CPU, memory and mass storage continues to widen.
- Well-written programs exhibit a property called locality.
- Memory hierarchies based on caching close the gap by exploiting locality.