The Memory Hierarchy
Sept 29, 2006

Topics
- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy
Random-Access Memory (RAM)

Key features

- RAM is traditionally packaged as a chip.
- Basic storage unit is normally a cell (one bit per cell).
- Multiple RAM chips form a memory.

Static RAM (SRAM)

- Each cell stores a bit with a four or six-transistor circuit.
- Retains value indefinitely, as long as it is kept powered.
- Relatively insensitive to electrical noise (EMI), radiation, etc.
- Faster and more expensive than DRAM.

Dynamic RAM (DRAM)

- Each cell stores bit with a capacitor. One transistor is used for access.
- Value must be refreshed every 10-100 ms.
- More sensitive to disturbances (EMI, radiation,…) than SRAM.
- Slower and cheaper than SRAM.
## SRAM vs DRAM Summary

<table>
<thead>
<tr>
<th></th>
<th>Tran. per bit</th>
<th>Access time</th>
<th>Needs refresh?</th>
<th>Needs EDC?</th>
<th>Cost</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>4 or 6</td>
<td>1X</td>
<td>No</td>
<td>Maybe</td>
<td>100x</td>
<td>cache memories</td>
</tr>
<tr>
<td>DRAM</td>
<td>1</td>
<td>10X</td>
<td>Yes</td>
<td>Yes</td>
<td>1X</td>
<td>Main memories, frame buffers</td>
</tr>
</tbody>
</table>

*EDC: Error Detection and Correction*
Conventional DRAM Organization

**d x w DRAM:**
- dw total bits organized as d supercells of size w bits

![Diagram of 16 x 8 DRAM chip]
- Memory controller
- Internal row buffer
- Supercell (2,1) marked
- Address (addr) and data (8 bits)
- Connections to CPU
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

64 MB memory module consisting of eight 8Mx8 DRAMs

addr (row = i, col = j)

Memory controller

64-bit doubleword at main memory address A

...
Enhanced DRAMs

DRAM Cores with better interface logic and faster I/O:

- Synchronous DRAM (SDRAM)
  Uses a conventional clock signal instead of asynchronous control
- Double data-rate synchronous DRAM (DDR SDRAM)
  Double edge clocking sends two bits per cycle per pin
- RamBus™ DRAM (RDRAM)
  Uses faster signaling over fewer wires (source directed clocking)
  with a Transaction oriented interface protocol

Obsolete Technologies:

- Fast page mode DRAM (FPM DRAM)
  Allowed re-use of row-addresses
- Extended data out DRAM (EDO DRAM)
  Enhanced FPM DRAM with more closely spaced CAS signals.
- Video RAM (VRAM)
  Dual ported FPM DRAM with a second, concurrent, serial interface
- Extra functionality DRAMS (CDRAM, GDRAM)
  Added SRAM (CDRAM) and support for graphics operations (GDRAM)
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
- Magnetic RAM (MRAM): stores bit magnetically (in development)
- Ferro-electric RAM (FERAM): uses a ferro-electric dielectric
- Programmable ROM (PROM): can be programmed once
- Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
- Electrically eraseable PROM (EEPROM): electronic erase capability
- Flash memory: EEPROMs with partial (sector) erase capability

Uses for Nonvolatile Memories
- Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,...)
- Solid state disks (flash cards, memory sticks, etc.)
- Smart cards, embedded systems, appliances
- Disk caches
Traditional Bus Structure Connecting CPU and Memory

A **bus** is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.
Memory Read Transaction (1)

CPU places address A on the memory bus.

Load operation: `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`
CPU read word x from the bus and copies it into register %eax.

Load operation: `movl A, %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}

\begin{itemize}
  \item Register file: \%eax
  \item ALU
  \item Bus interface
  \item I/O bridge
  \item Main memory
\end{itemize}
Memory Write Transaction (2)

CPU places data word y on the bus.

Store operation: `movl %eax, A`

```
%eax
  y
  ALU
  register file

bus interface

I/O bridge

y

main memory

0

A
```
Memory Write Transaction (3)

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

Store operation: `movl %eax, A`
Memory Subsystem Trends

Observation: A DRAM chip has an access time of about 50ns. Traditional systems may need 3x longer to get the data from memory into a CPU register.

- Modern systems integrate the memory controller onto the CPU chip: Latency matters!

- DRAM and SRAM densities increase and so does the soft-error rate:
  - Traditional error detection & correction (EDC) is a must have (64bit of data plus 8bits of redundancy allow any 1 bit error to be corrected and any 2 bit error is guaranteed to be detected)
  - EDC is increasingly needed for SRAMs too
  - ChipKill™ capability (can correct all bits supplied by one failing memory chip) will become standard soon
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**.
Aligned tracks form a cylinder.
Disk Capacity

Capacity: maximum number of bits that can be stored.
- Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9 Bytes (Lawsuit pending! Claims deceptive advertising).

Capacity is determined by these technology factors:
- Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
- Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
- Areal density (bits/in^2): product of recording and track density.

Modern disks partition tracks into disjoint subsets called recording zones
- Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
- Each zone has a different number of sectors/track
Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x
          (# tracks/surface) x (# surfaces/platter) x
          (# platters/disk)

Example:

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

Capacity = 512 x 300 x 20000 x 2 x 5
          = 30,720,000,000
          = 30.72 GB
Disk Operation (Single-Platter View)

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

arm

spindle
Disk Access Time

Average time to access some target sector approximated by:

- $T_{access} = T_{avg \ seek} + T_{avg \ rotation} + T_{avg \ transfer}$

Seek time ($T_{avg \ seek}$)
- Time to position heads over cylinder containing target sector.
- Typical $T_{avg \ seek} = 9$ ms

Rotational latency ($T_{avg \ rotation}$)
- Time waiting for first bit of target sector to pass under r/w head.
- $T_{avg \ rotation} = \frac{1}{2} \times \frac{1}{RPMs} \times 60$ sec/min

Transfer time ($T_{avg \ transfer}$)
- Time to read the bits in the target sector.
- $T_{avg \ transfer} = \frac{1}{RPM} \times \frac{1}{(avg \ # \ sectors/track)} \times 60$ secs/min.
Disk Access Time Example

Given:
- Rotational rate = 7,200 RPM
- Average seek time = 9 ms.
- Avg # sectors/track = 400.

Derived:
- Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
- Tavg transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
- Taccess = 9 ms + 4 ms + 0.02 ms

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.
Logical Disk Blocks

Modern disks present a simpler abstract view of the complex sector geometry:

- The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)

Mapping between logical blocks and actual (physical) sectors

- Maintained by hardware/firmware device called disk controller.
- Converts requests for logical blocks into (surface,track,sector) triples.

Allows controller to set aside spare cylinders for each zone.

- Accounts for the difference in “formatted capacity” and “maximum capacity”.
I/O Bus

The I/O bus is a communication pathway in a computer system that connects the CPU chip, memory, and various peripherals. It includes:

- **CPU chip**: Contains the ALU and register file.
- **Register file**: Stores data for the ALU operations.
- **ALU**: Performs arithmetic and logic operations.
- **Bus interface**: Connects the CPU to the I/O bridge.
- **I/O bridge**: Connects the system bus to the memory bus.
- **Main memory**: Stores data and instructions.
- **System bus**: Connects the CPU to the I/O bus.
- **Memory bus**: Connects the main memory to the I/O bus.
- **Expansion slots**: For other devices such as network adapters.
- **USB controller**: Manages USB devices.
- **Graphics adapter**: Manages graphics cards and monitors.
- **Disk controller**: Manages disk drives.
- **Mouse/keyboard**: Input devices.
- **Monitor**: Output device for visual display.
- **Disk**: Storage device for data and programs.

The I/O bus facilitates communication between the CPU and other components, enabling the system to interact with external devices and execute instructions from memory.
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)
# Storage Trends

## SRAM

<table>
<thead>
<tr>
<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>256</td>
</tr>
<tr>
<td>access (ns)</td>
<td>300</td>
<td>150</td>
<td>35</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>30</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>1,000</td>
<td>15,000</td>
</tr>
</tbody>
</table>

## DRAM

<table>
<thead>
<tr>
<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>0.20</td>
<td>40,000</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>8</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>1,000</td>
<td>15,000</td>
</tr>
</tbody>
</table>

## Disk

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB$</td>
<td>500</td>
<td>100</td>
<td>8</td>
<td>0.30</td>
<td>0.05</td>
<td>0.001</td>
<td>10,000</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>22</td>
</tr>
<tr>
<td>typical size(MB)</td>
<td>1</td>
<td>10</td>
<td>160</td>
<td>1,000</td>
<td>9,000</td>
<td>400,000</td>
<td>400,000</td>
</tr>
</tbody>
</table>
# CPU Clock Rates

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>8080</td>
<td>286</td>
<td>386</td>
<td>Pentium</td>
<td>P-III</td>
<td>P-4</td>
<td></td>
</tr>
<tr>
<td>clock rate(MHz)</td>
<td>1</td>
<td>6</td>
<td>20</td>
<td>150</td>
<td>750</td>
<td>3,000</td>
<td>3,000</td>
</tr>
<tr>
<td>cycle time(ns)</td>
<td>1,000</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.3</td>
<td>0.3</td>
<td>3,333</td>
</tr>
</tbody>
</table>
The CPU-Memory Gap

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

Principle of Locality:

- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.

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

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

Locality Example:

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference \( \text{sum} \) each iteration: **Temporal locality**

- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

```c
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```
Locality Example

**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?

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

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

    return sum;
}
```
Locality Example

Question: Does this function have good locality?

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

    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 and enduring properties of hardware and software:

- 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.
- Well-written programs tend to exhibit good locality.

These fundamental properties complement each other beautifully.

They suggest an approach for organizing memory and storage systems known as a memory hierarchy.
An Example Memory Hierarchy

- **L0:** registers
  - CPU registers hold words retrieved from L1 cache.

- **L1:** on-chip L1 cache (SRAM)
  - L1 cache holds cache lines retrieved from the L2 cache memory.

- **L2:** off-chip L2 cache (SRAM)
  - L2 cache holds cache lines retrieved from main memory.

- **L3:** main memory (DRAM)
  - Main memory holds disk blocks retrieved from local disks.

- **L4:** local secondary storage (local disks)
  - Local disks hold files retrieved from disks on remote network servers.

- **L5:** remote secondary storage (tapes, distributed file systems, Web servers)

- **Smaller, faster, and costlier (per byte) storage devices**
- **Larger, slower, and cheaper (per byte) storage devices**
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?
- 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.
- Net effect: A large pool of memory 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.
Caching in a Memory Hierarchy

Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1.

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

Data is copied between levels in block-sized transfer units.
Program needs object d, which is stored in some block b.

Cache hit
- Program finds b in the cache at level k. E.g., block 14.

Cache miss
- b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12.
- If level k cache is full, then some current block must be replaced (evicted). Which one is the “victim”?  
  - Placement policy: where can the new block go? E.g., b mod 4
  - Replacement policy: which block should be evicted? E.g., LRU
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+1.
  - 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.
# 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-byte 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>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>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>
Summary

- The memory hierarchy is fundamental consequence of maintaining the *random access memory* abstraction and practical limits on cost and power consumption.

- Caching works!

- Programming for good *temporal* and *spatial* locality is critical for high performance.

- Trend: the speed gap between CPU, memory and mass storage continues to widen, thus leading towards deeper hierarchies.
  - Consequence: maintaining locality becomes even more important.