Memory
Topics

- Memory hierarchy overview
- Storage technologies
- I/O architecture
- Storage trends
- Latency comparisons
- Locality
• Until now, we've referred to “memory” as a black box
• Modern systems actually have a variety of memory types called a memory hierarchy
  – Frequently-accessed data in faster memory
  – Each level caches data from the next lower level
• Goal: large general pool of memory that performs almost as well as if it was all made of the fastest memory
• Key concept: locality of time and space
• Other useful distinctions:
  – Volatile vs. non-volatile
  – Random access vs sequential access
Memory hierarchy

- **L0:** CPU registers hold words retrieved from cache memory.
- **L1:** L1 cache holds cache lines retrieved from the L2 cache.
- **L2:** L2 cache holds cache lines retrieved from L3 cache.
- **L3:** L3 cache holds cache lines retrieved from memory.
- **L4:** Main memory holds disk blocks retrieved from local disks.
- **L5:** Local disks hold files retrieved from disks on remote network servers.
- **L6:** Remote secondary storage (distributed file systems, Web servers).
History

- Delay-line memory (volatile, sequential)
- Magnetic core memory (non-volatile, random-access)

Images from Wikipedia
• Random Access Memory
  - Smaller pools of fast memory, closer to the CPU
  - **Volatile**: eventually lose data if the supply voltage is turned off
  - **Static RAM (SRAM)**
    • Six transistors per bit in a circuit w/ feedback loops
    • Essentially the same as discussed in Ch. 4
    • Used for CPU caches; usually <1GB
  - **Dynamic RAM (DRAM)**
    • One capacitor per bit with a single access transistor
    • Must be refreshed periodically
    • Used for main memory and graphics memory
    • Usually <64 GB
• DRAM chips store data in a grid of **supercells**
• **Memory controller** used to access data
  - Connected to CPU via memory bus
  - **Row access strobe (RAS)** request loads a row into a buffer
  - **Column access strobe (CAS)** request reads a particular supercell
Enhanced DRAM

- **Fast page mode DRAM** (FPM DRAM)
  - Serve same-row accesses from a row buffer
- **Extended data out DRAM** (EDO DRAM)
  - Allow CAS signals to be more closely spaced
- **Synchronous DRAM** (SDRAM)
  - Use a clock to synchronize and speed accesses
- **Double data-rate SDRAM** (DDR SDRAM)
  - Use both rising and falling edges of clock signal
- **Video RAM** (VRAM)
  - Shift an entire buffer's contents in a single operation
  - Allow simultaneous reads and writes
Nonvolatile memory

- **Nonvolatile memory** retains data if the supply voltage is turned off
  - Historically referred to as *read-only memory* (ROM)
  - Newer forms of nonvolatile memory can be written
- **Programmable ROM** (PROM)
  - Programmed only once by blowing fuses
- **Erasable PROM** (EPROM)
  - Re-programmed using ultraviolet light
- **Electrically-erasable PROM** (EEPROM)
  - Re-programmed using electric signals
  - Basis for *flash memory* storage devices
Non-volatile SRAM

- **Battery-backed SRAM (BBSRAM)**
  - External battery maintains value when power is off
- **Non-volatile SRAM (nvSRAM)**
  - Handles reads and writes the same as SRAM
  - Non-volatile component for permanent storage
  - Capacitor provides energy to store if current is lost
Disk storage

- Disk storage systems hold large amounts of data
  - More cost effective than SRAM or DRAM
  - Usually order of magnitudes slower
- Solid-state drives (SSDs)
  - Flash memory organized into blocks
- Traditional magnetic hard disk drives (HDDs)
  - Multiple platters with surfaces coated with magnetic material
  - Accessed using a physical arm with a magnetic head
  - Data stored on surface in tracks partitioned into sectors
Hard disk drives

- Capacity is based on **areal density**
  - Product of **recording density** and **track density**
- Operation requires mechanical motion
  - Magnetic read/write head on an **actuator arm**
- Speed is based on average **access time**
  - Sum of **seek time**, **rotational latency**, and **transfer time**
  - Platters spin at standard rate in one direction
- **Disk controller** coordinates accesses
  - Maps **logical blocks** to (surface, track, sector) numbers
Tape and network storage

- **Archival** storage systems provide large-scale data storage
  - Lowest cost per byte, but slowest access
- **Tape drives** store data on magnetic tape
  - Often in an off-site location for added redundancy
- **Network-attached storage (NAS) systems**
  - Dedicated data storage server
  - Often uses redundant disks for reliability (RAID)
  - Communicate over a network via a file sharing protocol
  - Examples: NFS, Samba, AFS
  - *More about this in CS 361 and CS 470!*
I/O architecture

Diagram showing different components and levels of the I/O architecture:

- **L0**: Register file and ALU
- **L1-L3**: Level 1-3 cache
- **L4**: Main memory
- **L5**: Disk controller and drive
- **L6 (NAS)**: Expansion slots for other devices such as network adapters

The diagram also includes peripherals like USB controllers, graphics adapters, and host bus adapters (SCSI/SATA) connected to the I/O bus.
I/O architecture

- **Registers and cache memory (SRAM)**
  - Access via direct connection to CPU (or on-die)
- **Main memory (DRAM)**
  - Bus transactions via I/O bridge on motherboard
- **Disk drives (magnetic disk & SSD)**
  - Connected to I/O bridge via I/O bus
  - Requires a device controller for communication
  - Memory transactions w/o CPU via direct memory access (DMA)
  - Technologies: USB, SATA, SCSI
- **Other memory (graphics, network storage)**
  - Connected to I/O bus using expansion slots on motherboard
1) CPU initiates disk read
2) Disk reads data
3) Disk writes RAM via DMA
4) Disk notifies CPU

This is much faster than reading data from disk into registers then storing it in memory, and the CPU can do other tasks while this happens.
Technology comparison

- Disk seek time
- SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time

(w/ multicore CPUs)
## Storage trends

<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>2,900</td>
<td>320</td>
<td>256</td>
<td>100</td>
<td>75</td>
<td>60</td>
<td>25</td>
<td>116</td>
</tr>
<tr>
<td>Access (ns)</td>
<td>150</td>
<td>35</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>1.3</td>
<td>115</td>
</tr>
</tbody>
</table>

(a) SRAM trends

<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>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>0.1</td>
<td>0.06</td>
<td>0.02</td>
<td>44,000</td>
</tr>
<tr>
<td>Access (ns)</td>
<td>200</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Typical size (MB)</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2,000</td>
<td>8,000</td>
<td>16,000</td>
<td>62,500</td>
</tr>
</tbody>
</table>

(b) DRAM trends

<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>$/GB</td>
<td>100,000</td>
<td>8,000</td>
<td>300</td>
<td>10</td>
<td>5</td>
<td>0.3</td>
<td>0.03</td>
<td>3,333,333</td>
</tr>
<tr>
<td>Min. seek time (ms)</td>
<td>75</td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>Typical size (GB)</td>
<td>0.01</td>
<td>0.16</td>
<td>1</td>
<td>20</td>
<td>160</td>
<td>1,500</td>
<td>3,000</td>
<td>300,000</td>
</tr>
</tbody>
</table>

(c) Rotating disk trends

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel CPU</td>
<td>80286</td>
<td>80386</td>
<td>Pent.</td>
<td>Pent. 4</td>
<td>Core 2</td>
<td>Core i7 (n)</td>
<td>Core i7 (h)</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Clock rate (MHz)</td>
<td>6</td>
<td>20</td>
<td>150</td>
<td>600</td>
<td>3,300</td>
<td>2,000</td>
<td>2,500</td>
<td>3,000</td>
<td>500</td>
</tr>
<tr>
<td>Cycle time (ns)</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.5</td>
<td>0.4</td>
<td>0.33</td>
<td>500</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Effective cycle time (ns)</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.30</td>
<td>0.25</td>
<td>0.10</td>
<td>0.08</td>
<td>2,075</td>
</tr>
</tbody>
</table>

(d) CPU trends

---

Clock rates and cycle times have stalled, but effective cycle times continue to decrease.
# Latency Numbers Every Programmer Should Know

<table>
<thead>
<tr>
<th>Latency Comparison</th>
<th>Main Memory Reference: 100ns</th>
<th>Send 1KB over 1Gbps Network: 10μs</th>
<th>Read 1MB Sequentially from SSD: 1 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ns</td>
<td>1μs</td>
<td>10μs</td>
<td>Disk Seek: 10μs</td>
</tr>
<tr>
<td>L1 cache reference: 0.5 ns</td>
<td></td>
<td>SSD random read (10μs SSD): 150μs</td>
<td>Read 1MB Sequentially from disk: 20μs</td>
</tr>
<tr>
<td>Branch mispredict: 5 ns</td>
<td></td>
<td>Read 1MB Sequentially from memory: 250μs</td>
<td>Packet roundtrip CR to Netherlands: 150μs</td>
</tr>
<tr>
<td>L2 cache reference: 7 ns</td>
<td></td>
<td>Round trip in same datacenter: 500μs</td>
<td></td>
</tr>
<tr>
<td>Mutex lock/unlock: 25 ns</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: https://gist.github.com/2841832
Latency comparison

Lets multiply all these durations by a billion: (originally from https://dzone.com/articles/every-programmer-should-know)

**Minute:**
- L1 cache reference (0.5s) - *One heart beat*
- L2 cache reference (7s) - *Long yawn*

**Hour:**
- Main memory reference (100s) - *Brushing your teeth*

**Day:**
- Send 2K bytes over 1 Gbps network (5.5 hr) - *From lunch to end of work day*

**Week:**
- SSD random read (1.7 days) - *A normal weekend*
- Read 1 MB sequentially from memory (2.9 days) - *A long weekend*
- Read 1 MB sequentially from SSD (11.6 days) - *Waiting for almost 2 weeks for a delivery*

**Year:**
- Disk seek (16.5 weeks) - *A semester in university*
- Read 1 MB sequentially from disk (7.8 months) – *Two semesters in university*
- The above 2 together (1 year)

**Decade:**
- Send packet CA->Netherlands->CA (4.8 years) - *Completing a bachelor's degree*
Locality

• **Temporal locality**: frequently-accessed items will continue to be accessed in the future
  - Theme: *repetition is common*

• **Spatial locality**: nearby addresses are more likely to be accessed soon
  - Theme: *sequential access is common*

• Why do we care?
  - *Programs with good locality run faster than programs with poor locality*
Data locality

• Using predictable access patterns exhibits *spatial* locality
  - Stride-1 reference pattern (sequential access)
  - Stride-k reference pattern (every k elements)
  - Closely related to *row-major* vs. *column-major*
  - Allows for *prefetching* (predicting the next needed element and preloading it)

• Re-using values many times exhibits *temporal* locality
  - Can keep them in a high level of the memory hierarchy
Instruction locality

• Normal execution exhibits *spatial* locality
  – Instructions execute **in sequence**
  – Long jumps exhibit poor locality (this includes calls)
• Loops exhibit both *temporal* and *spatial* locality
  – The body statements execute **repeatedly** (temporal locality) and **in sequence** (spatial locality)
  – Short loops are better
int64_t nums[4] = {
    0x12, 0x23,
    0x34, 0x45
};

for (size_t i = 0; i < 4; i++) {
    nums[i] += 1;
}

What is the sequence of memory accesses for this code?
Core themes

• Systems design involves tradeoffs
  – Memory: price vs. performance (e.g., DRAM vs. SRAM)

• The details matter!
  – Knowledge of the underlying system enables you to exploit latency inequalities for better performance

• Key concepts: locality and caching
  – Store and access related things together
  – Keep copies of things you’ll need again soon
  – We’ll look at these more next time