CMSC 611: Advanced Computer Architecture

Memory, I/O and Disk

Most slides adapted from David Patterson. Some from Mohomed Younis.
Main Memory Background

- Performance of Main Memory:
  - Latency: affects cache miss penalty
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  - Bandwidth: primary concern for I/O & large block

- Main Memory is DRAM: Dynamic RAM
  - Dynamic since needs to be refreshed periodically
  - Addresses divided into 2 halves (Row/Column)

- Cache uses SRAM: Static RAM
  - No refresh
    - 6 transistors/bit vs. 1 transistor/bit, 10X area
  - Address not divided: Full address
4 Mbit DRAM: square root of bits per RAS/CAS

- Refreshing prevent access to the DRAM (typically 1-5% of the time)
- Reading one byte refreshes the entire row
- Read is destructive and thus data need to be re-written after reading
  - Cycle time is significantly larger than access time
**Processor-Memory Performance**

CPU-DRAM Gap “Moore’s Law”

- CPU: 60%/yr. (2X/1.5yr)
- DRAM: 9%/yr. (2X/10 yrs)

**Problem:**
Improvements in access time are not enough to catch up

**Solution:**
Increase the bandwidth of main memory (improve throughput)
Memory Organization

a. One-word-wide memory organization

• **Simple**: CPU, Cache, Bus, Memory same width (32 bits)
• **Wide**: CPU/Mux 1 word; Mux/Cache, Bus, Memory N words
• **Interleaved**: CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*

b. Wide memory organization

Memory organization would have significant effect on bandwidth
Memory Interleaving

- Access Pattern without Interleaving:

- Access Pattern with 4-way Interleaving:

We can Access Bank 0 again
Independent Memory Banks

• Original motivation: sequential access
• Multiple independent accesses:
  – Multiprocessor system / concurrent execution
  – I/O: limiting memory access contention
  – CPU with Hit under n Misses, Non-blocking Cache
• Multiple access requires per-bank
  – controller, address bus and possibly data bus
  
<table>
<thead>
<tr>
<th>Superbank number</th>
<th>Superbank offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank number</td>
<td>Bank offset</td>
</tr>
</tbody>
</table>

  – **Superbank**: all memory active on one block transfer
  – **Bank**: portion within a superbank that is word interleaved (or Subbank)

Superbanks act as separate memories mapped to the same address space
Avoiding Bank Conflicts

• The effectiveness of interleaving depends on the frequency that independent requests will go to different banks
  – Bank number = address % number of banks
  – Address within bank = address / words in bank

• Example: Assuming 128 banks
  
  ```
  int x[256][512];
  for (j = 0; j < 512; j = j+1)
      for (i = 0; i < 256; i = i+1)
          x[i][j] = 2 * x[i][j];
  ```

• Since 512 is a multiple of 128
  – Every column in same bank
Avoiding Bank Conflicts

• Solutions:
  – SW: loop interchange or declaring array not power of 2 (“array padding”)
  – HW: Prime number of banks and modulo interleaving
    • Complexity of modulo & divide per memory access with prime no. banks?
    • Simple address calculation using the Chinese Remainder Theorem
Chinese Remainder Theorem

• As long as two sets of integers $a_i$ and $b_i$ follow these rules:
  – $b_i = x \mod a_i$
  – $0 \leq b_i < a_i$
  – $0 \leq x < a_0 \times a_1 \times a_2 \times \ldots$
  – $a_i$ and $a_j$ are co-prime with $i \neq j$

• then the integer $x$ has only one solution
Apply to Bank Addressing

• Modulo interleaving
  – Bank number = $b_0$, Number of banks = $a_0$
  – Address in bank = $b_1$, Words in bank = $a_1$
  – Bank number < number of banks \( (0 \leq b_0 < a_0) \)
  – Address in bank < words in a bank \( (0 \leq b_1 < a_1) \)
  – Address < Number of banks \( \times \) words in bank
    • \( (0 \leq x < a_0 \times a_1) \)
  – The number of banks \( (a_0) \) and words in a bank are co-prime \( (a_1) \)
    • $a_1$ a power of 2, $a_0$ prime > 2
Example

- Bank number = address MOD number of banks
- Address within bank = address MOD words in bank
- Bank number = \( b_0 \), number of banks = \( a_0 \) (= 3 in example)
- Address within bank = \( b_1 \), number of words in bank = \( a_1 \) (= 8 in example)

<table>
<thead>
<tr>
<th>Bank Number:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq. Interleaved</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Address within Bank:</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

Unambiguous mapping with simple bank addressing
DRAM-specific Optimization

• DRAM Access Interleaving
  – DRAM must buffer a row of bits internally for the column access
    • Performance can be improved by allowing repeated access to buffer with our another row access time (requires minimal additional cost)
  – Page mode: The buffer acts like a SRAM allowing bit access from the buffer until a row change or a refresh
  – Static column: Similar to page mode but does not require change in CAS to access another bit from the buffer
  – DRAM optimization has been shown to give up to 4x speedup
• Bus-based DRAM (RAMBUS)
  – Each chip act as a module with an internal bus replacing CAS and RAS
  – Allows for other accesses to take place between the sending the address and returning the data
  – Each module performs its own refresh
  – Performance can reach 1 byte / 2 ns
    • (500 MB/s per chip)
  – Expensive compared to traditional DRAM
• I/O Interface
  – Device drivers
  – Device controller
  – Service queues
  – Interrupt handling

• Design Issues
  – Performance
  – Expandability
  – Standardization
  – Resilience to failure

• Impact on Tasks
  – Blocking conditions
  – Priority inversion
  – Access ordering
Impact of I/O on System Performance

Suppose we have a benchmark that executes in 100 seconds of elapsed time, where 90 seconds is CPU time and the rest is I/O time. If the CPU time improves by 50% per year for the next five years but I/O time does not improve, how much faster will our program run at the end of the five years?

**Answer:**

\[
\text{Elapsed Time} = \text{CPU time} + \text{I/O time}
\]

<table>
<thead>
<tr>
<th>After n years</th>
<th>CPU time</th>
<th>I/O time</th>
<th>Elapsed time</th>
<th>% I/O time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90 Seconds</td>
<td>10 Seconds</td>
<td>100 Seconds</td>
<td>10%</td>
</tr>
<tr>
<td>1</td>
<td>(\frac{90}{1.5} = 60) Seconds</td>
<td>10 Seconds</td>
<td>70 Seconds</td>
<td>14%</td>
</tr>
<tr>
<td>2</td>
<td>(\frac{60}{1.5} = 40) Seconds</td>
<td>10 Seconds</td>
<td>50 Seconds</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>(\frac{40}{1.5} = 27) Seconds</td>
<td>10 Seconds</td>
<td>37 Seconds</td>
<td>27%</td>
</tr>
<tr>
<td>4</td>
<td>(\frac{27}{1.5} = 18) Seconds</td>
<td>10 Seconds</td>
<td>28 Seconds</td>
<td>36%</td>
</tr>
<tr>
<td>5</td>
<td>(\frac{18}{1.5} = 12) Seconds</td>
<td>10 Seconds</td>
<td>22 Seconds</td>
<td>45%</td>
</tr>
</tbody>
</table>

**Over five years:**

CPU improvement = \(\frac{90}{12} = 7.5\) \hspace{1cm} **BUT** \hspace{1cm} System improvement = \(\frac{100}{22} = 4.5\)
Typical I/O System

- The connection between the I/O devices, processor, and memory are usually called (local or internal) bus
- Communication among the devices and the processor use both protocols on the bus and interrupts
## I/O Device Examples

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (KB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Line Printer</td>
<td>Output</td>
<td>Human</td>
<td>1.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Optical Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>500.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>5,000.00</td>
</tr>
<tr>
<td>Network-LAN</td>
<td>Input or Output</td>
<td>Machine</td>
<td>20 – 1,000.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
</tbody>
</table>
Storage Technology Drivers

• Driven by the prevailing computing paradigm:
  – 1950s: migration from batch to on-line processing
  – 1990s: migration to ubiquitous computing
    • Computers in phones, books, cars, video cameras, …
    • Nationwide fiber optical network with wireless tails

• Effects on storage industry:
  – Embedded storage: smaller, cheaper, more reliable, lower power
  – Data utilities: high capacity, hierarchically managed storage
Disk History

Data density in Mbit/square inch

Capacity of Unit Shown in Megabytes

Magnetic Disk

- **Purpose:**
  - Long term, nonvolatile storage
  - Large, inexpensive, and slow
  - Low level in the memory hierarchy

- **Characteristics:**
  - Rely on rotating platters coated with a magnetic surface
  - Use a moveable read/write head to access the disk
  - Platters are rigid (metal or glass)
Organization of a Hard Magnetic Disk

- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
    - A sector is the smallest unit that can be read or written to
- Traditionally all tracks have the same number of sectors:
  - Constant bit density: record more sectors on the outer tracks
  - Recently relaxed: constant bit size, speed varies with track location