I/O

Who cares about I/O?
Latency & throughput
I/O characteristics
Device characteristics and types
Review of important devices: disks, graphics displays, networks
I/O system architecture: buses, I/O processors
High performance disk architectures

Why I/O?

Amdahl's law
✓ if you only speed up CPU then I/O becomes the bottleneck
e.g.,
✓ suppose I/O takes 10% of the time
✓ speed up CPU 10 times
✓ system only speeds up by 5 times

Latency vs. Throughput

“There is an old saying: bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed. You can’t bribe God.” - David Clark

Throughput:
✓ bandwidth
✓ I/Os per second
Latency:
✓ response time
Latency vs. Throughput

Who cares about latency?
- Why don't you just context switch?
- Fallacy
  - requires more memory
  - requires more processes
  - human productivity increases super-linearly with response time

I/O Overlap

I/O overlaps with computation in complicated ways

I/O Performance

\[ \text{Time}_{\text{job}} = \text{time}_{\text{cpu}} + \text{time}_{\text{IO}} - \text{time}_{\text{overlap}} \]

e.g.,

\[ 10 = 10 + 4 - 4 \]

Speed up CPU by 2x

What is \( \text{time}_{\text{job}} \)?

- \( \text{time}_{\text{job}} = 5 + 4 - 4 = 5 \) (best)
- \( \text{time}_{\text{job}} = 5 + 4 - 0 = 9 \) (worst)
- \( \text{time}_{\text{job}} = 5 + 4 - 2 = 7 \) (average?)

I/O Characteristics

Supercomputers
- data transfer rate important
- many MBs per second for large files

Transaction processing
- I/O rate important
- "random" accesses
- disk I/Os per second is important
**I/O Characteristics**

- Time sharing filesystems
- □ small files
- □ sequential accesses
- □ many creates/deletes

---

**Device Characteristics**

- **Behavior**
  - □ input - read once
  - □ output - write once
  - □ storage - read many times, usually write

- **Partner**
  - □ human
  - □ machine

- **Data rate**
  - □ peak transfer rate

---

**Device Characteristics**

<table>
<thead>
<tr>
<th>Device</th>
<th>I or O?</th>
<th>Partner</th>
<th>Data Rate KB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse</td>
<td>I</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Graphics display</td>
<td>O</td>
<td>Human</td>
<td>60,000</td>
</tr>
<tr>
<td>Modem</td>
<td>I/O</td>
<td>Machine</td>
<td>2-8</td>
</tr>
<tr>
<td>LAN</td>
<td>I/O</td>
<td>Machine</td>
<td>500-6000</td>
</tr>
<tr>
<td>Tape</td>
<td>Storage</td>
<td>Machine</td>
<td>2000</td>
</tr>
<tr>
<td>Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>2000-10000</td>
</tr>
</tbody>
</table>

---

**Magnetic Disks**

- Head
- Arm
- Platters
- Cylinder Spans
- Track
- Sector
- Intersector gap
**Disk Parameters**

- Spindles: 1-12 (most 1)
- Surfaces per spindle: 1-24
- rpm: 3600-10,000
- Platter diameter: 1"-8"
  - trend towards smaller disks
  - higher RPM
  - mass production
- Tracks per surface: 5000-20,000

**Disk Parameters (Cont.)**

- Sectors per surface: about 100 to about 400
  - sector # -- gap -- data + ECC
  - typically fixed length sectors
  - variable # of sectors per track
  - outer track is longer than inner track
  - so-called “constant bit density”

**Disk Operations**

- Seek: move head to track
  - avg seek time = \[ \sum_{i=1}^{n} \text{seek}(i) / n \]
  - n is # tracks, seek(i) is time to seek ith track
- Rotational latency: wait for sector to come under head
  - avg rotational latency 0.5/3600 = 8.3 ms
  - avg rotational latency 0.5/10000 = 3 ms
- Transfer: move data to/from disk
  - typically 3-30 MB per second
  - function of bits/inch of track and rotation rate

**Disk Operations**

- Overhead
  - controller delay
  - queuing delay
**Disk Performance**

Avg disk access = avg seek time + avg rot. Delay + transfer + ovhd
e.g.,
- 3600 rpm, 2 MB/s
- avg seek time: 9ms
- controller overhead: 1ms
- read 512-byte sector
- 9ms + 0.5/3600 + 0.5KB/2 + 1 = 18.6ms
- effective bandwidth: 0.5KB / 18.6 ms = 27KB/s!

**Improving Disk Performance**

Use large sectors to improve bandwidth
Use track caches & read ahead
- read entire track into on-controller cache
- exploit locality (improves both latency and bandwidth)

Design file systems to maximize locality
- allocate files sequentially on disk (exploit track cache)
- locate similar files in same cylinder (reduce seeks)
- locate similar files in near-by cylinders (reduce seek distance)

Pack bits closer together to improve transfer rate (and density)

**Improving Disk Density**

Areal density = tracks/inch * bits/inch of track
- increasing density → bits closer together
- greater likelihood of bit errors

Error Correction Codes can detect and correct bit errors
- standard SECDED codes correct single bit errors
- stronger codes can correct burst errors
  - e.g., Reed-Solomon codes

Key Idea: Use ECC to improve density
- pack bits more tightly (linear benefit)
- add strong ECC to correct errors (logarithmic cost)
- significant potential improvement

**Alternatives to Disks**

DRAMS
- SSD (solid-state disk)
  - standard disk interface
  - DRAM and battery backup
- ES (expanded storage)
  - software controlled cache
  - large (4K) blocks
- no seek time
- fast transfer rate
- expensive
Alternatives to Disks (Cont.)

FLASH memory
- no seek time
- fast transfer
- non-volatile
- bulk erase before write
- slow writes
- “wears” out over time

Optical Disks

Read-only
- cheap & reliable
- slow

Write once
- almost as cheap
- slow

Write many
- more expensive
- slow

Graphics Display: CRT

Phosphorous-Coated Screen

Electron Gun

X + Y Deflectors

Screen has many scan lines each of which has many pixels
Phosphorous acts as capacitor: refresh 30-60 times/second

Graphics Display: Frame Buffer

CPU

Memory

Frame Buffer

CRT

0.2+ MB/s

30+ MB/s
Frame Buffer (Cont.)
Frame buffer stores bitmap
- one entry per pixel
  - black: 1 bit per pixel
  - gray-scale: 4-8 bits per pixel
  - color (RGB): 8 bits per color (24 bits)
Typical size 1560x1280 pixels
Black and white: 250KB
Color (24 bit RGB): 5.7 MB

Reducing Frame Buffer Cost
Key idea: only a small number of colors are used in one image
Color map:
- a table of pixel color values
- frame buffer stores table index instead

Frame Buffer Operations
Logically output only
- but read as well
BIT BLTS
- read-modify-write operations
- e.g., read xor write
- used for cursors, etc.
Open question
- OS only?
- Or direct user accesses?

Frame Buffer Implementation
1560x1280 RGB display
- bandwidth required = 1560x1280x24/8x30 = 171 MB/s
How can we implement this?
- Video DRAMs
- dual-ported DRAM
  - regular random access port
  - serial video port
- use 24 in parallel for RGB
What about bandwidth? Interleave video DRAMs
Simple graphics computation:
- iRAM: Intelligent RAM
- Vector computation within DRAM for Video
Other Issues in Displays

- Double buffering
  - duplicate frame buffer
  - to prevent displaying incomplete updates
  - may be necessary for animation
- Z-buffer
  - for displaying 3-D images
  - store z-dimension for each pixel
  - BIT BLTS compare z-dimension

I/O System Architecture

- Hierarchical data paths
  - divides bandwidth going down hierarchy
  - often buses at each level
- I/O processing
  - program controlled
  - DMA
  - dedicated I/O processors

I/O System Architecture

![I/O System Architecture Diagram]

Buses

<table>
<thead>
<tr>
<th>Option</th>
<th>High Performance</th>
<th>Low cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address/data lines</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data lines</td>
<td>Wider</td>
<td>Narrower</td>
</tr>
<tr>
<td>Transfer size</td>
<td>Multiple words</td>
<td>Single word</td>
</tr>
<tr>
<td>Bus masters</td>
<td>Multiple</td>
<td>One</td>
</tr>
<tr>
<td>Split transaction</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Clocking</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>
Buses

CPU-memory buses
- want speed
- usually custom design

I/O buses
- compatibility is important
- usually standard designs: PCI, SCSI

I/O to CPU Interface

Physical connection
1. Direct to cache
   - no coherence problems
   - DMA
   - may not be standard

2. CPU-memory bus
   - DMA
   - may not be standard

I/O bus
- industry standard
- slower than memory bus
- indirection through I/O processor
Bus Arbitration

Centralized star connection
- high cost
- high performance

Daisy chain
- cheap
- low performance

Distributed arbitration
- medium price/performance

Arbitration for next bus mastership overlap with current transfer

Distributed Arbitration

Set of wire-or priority lines
Set of wire-or timing and control lines
Each requesting device indicates its priority
Device removes its less-significant bits if higher priority present
Eventually only highest priority remains
Special care to ensure fairness

Bus Switching Methods

Circuit-switched buses
- held until request is complete
- simple protocol
- device latency affects bus utilization

Split-transaction or pipelined
- bus is released after request is initiated
- other of the bus until reply comes back
- Pipelined (packet-switched): reply comes back at a fixed time
- Split-transaction: reply must arbitrate
- complex bus protocol
- better utilization of bus

Standard I/O Buses

<table>
<thead>
<tr>
<th>Param</th>
<th>S-bus</th>
<th>Micro Channel</th>
<th>PCI</th>
<th>SCSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data width (bits)</td>
<td>32</td>
<td>32</td>
<td>32-64</td>
<td>8-16</td>
</tr>
<tr>
<td>Clock (Mhz)</td>
<td>16-25</td>
<td>Asynch Multiple</td>
<td>33 Multiple</td>
<td>10/async Multiple</td>
</tr>
<tr>
<td>#masters</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>B/w, 32-bit read (MB/s)</td>
<td>33</td>
<td>20</td>
<td>33</td>
<td>20 or 6</td>
</tr>
<tr>
<td>B/w, peak (MB/s)</td>
<td>89</td>
<td>75</td>
<td>111</td>
<td>20 or 6</td>
</tr>
</tbody>
</table>
Memory Buses

<table>
<thead>
<tr>
<th>Param</th>
<th>HP Summit</th>
<th>SGI Challenge</th>
<th>Sun XDBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data width (bits)</td>
<td>128</td>
<td>256</td>
<td>144</td>
</tr>
<tr>
<td>Clock (Mhz)</td>
<td>60</td>
<td>48</td>
<td>66</td>
</tr>
<tr>
<td>#masters</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>B/w, peak (MB/s)</td>
<td>960</td>
<td>1200</td>
<td>1056</td>
</tr>
</tbody>
</table>

Communicating I/O Processors

1. I/O control
   - Memory-mapped
     - ID/ID to aligned addresses => operations occur
     - protected by Virtual Memory
   - I/O-mapped
     - special instructions initiate I/O operations
     - protected by privileged instructions

2. I/O completion
   - Polling
     - wait for status bit to change, periodically check
   - Interrupts
     - completion interrupts CPU

IBM 3990 I/O Processing

Channel = IOP
1. User program sets up table in memory with I/O request
   - pointer to channel program
   - execute a syscall
2. OS checks for protection, then executes “start subchnl” instr
3. Pointer to channel program is passed to IOP.
   - IOP executes channel program
4. IOP interacts with storage director
   - executes individual channel commands
   - free to do other work between channel commands
5. On completion, IOP places status in memory, interrupts CPU

High-Performance Disk Architectures

Extension to conventional disks
Disk arrays
Redundant array of inexpensive disks (RAIDs)
Extensions to Conventional Disk Architectures

Fixed head disk
- head per track, head does not seek
- seek time eliminated
- rotational latency unchanged
- low track density
- not economical

Parallel transfer disk
- read from multiple surfaces at the same time
- difficulty in looking onto different tracks on multiple surfaces
- lower cost alternatives possible (disk arrays)

Increasing disk density
- an on-going process
- requires increasingly sophisticated lock-on control
- increases cost

Solid state disks

Disk caches
- RAM to buffer data between device and host
- fast writes: buffer acts as a write buffer
- better utilization of host-to-device path
- high miss rate increases request latency

Disk scheduling
- schedule simultaneous I/O requests to reduce latency
- e.g., schedule request with shortest seek time
- works best for unlikely cases (long queues)

Collection of individual disks
- each disk has its own arm/head

Data distributions
- Independent
- Fine-grain
- Coarse-grain
Disk Arrays (Cont.)

Independent addressing
- s/w user distribute data
- load balancing an issue

Fine-grain striping
- one bit, one byte, one sector
- disks x stripe unit evenly divides smallest accessible data
- perfect load balance; only one request served at a time
- effective transfer rate approx. N times better than single disk
- access time can go up, unless synchronized disks

Coarse-grain striping
- data transfer parallelism for large requests
- concurrency for small requests
- load balanced by statistical randomization

Must consider workload to determine stripe size

Redundancy Mechanisms

Disk failures are a significant fraction of hardware failures
- striping increases #corrupted files per failure

Data replication
- disk mirroring
- allow multiple reads
- writes must be synchronized

Parity protection
- use a parity disk

Redundant Array of Inexpensive Disks

RAID (Berkeley)
- arrays of small cheap disks provide high performance/reliability

D = #data disks, C = #check disks

Level1: mirrored disks (D = 1, C = 1)
- high overhead (100%)

Level2: bit interleaved array (e.g., D=10, C=4)
- like ECC for DRAMS
- read all bits across groups
- merge update bits with bits not updated
- recompute parity
- rewrite full group including checks

RAID (Cont.)

Level3: hard failure detection and parity (e.g., D = 4, C = 1)
- key: failed disk is easily identified by controller
- no need for special code to identify failed disk

Level4: intra-group parallelism
- coarse-grain striping
- like level-3 + ability to do more than one small I/O at a time
- write must update disk with data and parity disk
RAID (Cont.)

Level 5: rotated parity to parallelize writes
- parity spread out across disks in a group
- different updates of parities go to different disks

Level 6: two-dimensional array
- array of data is a two-dimensional
- with row and column parities

IBM Millipede

Nanotech storage technique
Atomic force probes