Don’t forget...

- Lectures on Graphics Processors (GPUs):
  - Thursday: 10-12, room 1211
    - Introduction to Graphics Processors
    - GPUs: The Hype, The Reality, and The Future
  - Friday: 10-12, room 1211
    - Introduction to OpenCL
    - OpenCL Case Study: Optimizing a PDE Solver

I/O Today

- Devices
  - Disks
  - Keyboard/Mouse
  - Network
  - Displays
- Interfaces
  - USB
  - PCIe
  - SATA
  - Ethernet
  - Wireless (802.11n, bluetooth)
  - Audio, video
  - FireWire
  - Thunderbolt

Interface Technologies

<table>
<thead>
<tr>
<th>Interface</th>
<th>Throughput</th>
<th>Power</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB</td>
<td>1.5, 60, 625 MB/s</td>
<td>12MB/s</td>
<td>480MB/s</td>
</tr>
<tr>
<td>PCIe/PCIe</td>
<td>133, 5, 120 MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SATA</td>
<td>180, 375, 750 MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td>1.25/2.5/6 GB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FireWire</td>
<td>5G, 100 MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thunderbolt</td>
<td>1.250 MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>802.11</td>
<td>6.75, 13, 6.75, 18.75 MB/s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Storage Technologies

<table>
<thead>
<tr>
<th>Storage Technology</th>
<th>Capacity</th>
<th>Longevity</th>
<th>Latency</th>
<th>Throughput</th>
<th>Power</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>8000GB</td>
<td>20 years</td>
<td>10 s</td>
<td>140 MB/s</td>
<td>$0.06/GB</td>
<td></td>
</tr>
<tr>
<td>Disk</td>
<td>3TB</td>
<td>3-5 years</td>
<td>8 ms</td>
<td>200 MB/s</td>
<td>$0.07/GB</td>
<td></td>
</tr>
<tr>
<td>NVRAM</td>
<td>256GB</td>
<td>10 years</td>
<td>80 µs</td>
<td>250 MB/s</td>
<td>$1.48/GB</td>
<td></td>
</tr>
<tr>
<td>SRAM</td>
<td>8MB</td>
<td>0</td>
<td>15 ms</td>
<td>26,624 MB/s</td>
<td>$7,200/GB</td>
<td></td>
</tr>
<tr>
<td>SRAM</td>
<td>32kB</td>
<td>0</td>
<td>1.3</td>
<td>47,104 MB/s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- New Technologies: PCRAM (phase-change), MRAM (magnetic)
- Open Question: NVRAM is new. How do we best use it?
  - Replacement for disks?
  - Expanded system memory?

Disks: What Do You Care About?

- HPC and media: throughput (total data per second)
- Database and web: latency (time to first data)
- Desktop: mixture
- Cell phone: ?

From: storagereview.com
Different Latencies Can Do Strange Things

Effect of turning on an SSD cache (much lower overall latency)

System level bandwidths

Latencies over time for a large server setup with multiple disks.

Busses

- Shared communication medium
  - Lots of devices have access
  - Lots of bits transferred at once
  - Run all wires to all devices
- Pros/Cons
  - Simple, everyone has access, easy to add devices
  - Hard to make wires go everywhere and still be fast
- Examples
  - Memory Busses
  - I/O busses

Bus Classifications

- CPU-memory busses
  - Fast
  - Proprietary
  - Closed and controlled
  - Support only memory transactions
- IO busses
  - Standardized (SCSI, PCI)
  - More diversity
  - More length
- Bus Bridges/Adapter
  - Cross from one bus to another

Bus Design Decisions

<table>
<thead>
<tr>
<th></th>
<th>High Performance</th>
<th>Low Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Split Address/Data</td>
<td>Multiplex Address/Data</td>
</tr>
<tr>
<td>Width</td>
<td>Wide</td>
<td>Narrow</td>
</tr>
<tr>
<td>Transfer Size</td>
<td>Large/Flexible</td>
<td>Small</td>
</tr>
<tr>
<td>Split Transactions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mastering</td>
<td>Multiple</td>
<td>Single</td>
</tr>
<tr>
<td>Clocking</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>

Bus Clocking: Synchronous

- Synchronous
  - Sample the control signals at edge of clock

Pro:
- Fast and High Performance

Con:
- Can’t be long (skew) or fast at same time
- All bus members must run at the right speed
**Bus Clocking: Asynchronous**

- Asynchronous
  - Edge of control signals determines communication
  - "Handshake Protocol"

1. Request (with actual transaction)
2. Acknowledge causes de-assert of Request
3. De-assert of Request causes de-assert of Ack
4. De-assert of Ack allows re-assertion of Request

**Structure, Width, and Transfer Length**

- Separate vs. Multiplexed Address/Data
  - Multiplexed: save wires
  - Separate: more performance
- Wide words: higher throughput, less control per transfer
  - On-chip cache to CPU buses: 256 bits wide
  - Serial Busses
- Data Transfer Length
  - More data per address/control transfer

- Example: Multiplexed Addr/Data with Data transfer of 4
  - Address overhead reduced to 16% = 20%

**Split Transactions**

Problem: Long wait times for response

Solution: Split Transactions

**Bus Mastering**

- Bus Master: a device that can initiate a bus transfer

- Example:
  1. CPU makes memory request
  2. Page Fault in VM requires disk access to load page
  3. Move data from disk to memory

If the CPU is master, does it have to check to see if the disk is ready to transfer?

**Multiple Bus Masters**

- What if multiple devices could initiate transfers?
  - Update might take place in background while CPU operates
- Multiple CPUs on shared memory systems
- Challenge: Arbitration
  - If two or more masters want the bus at the same time, who gets it?

**Arbitration Goals**

- Functionality
  - Prevent bus conflicts (two bussed simultaneous drivers)
- Performance
  - Need to make decisions quickly
- Priority
  - Some masters are more desperate than others
    - Example: DRAM refresh
- Fairness
  - Every equal priority master should get equal service
  - No "starvation": Every requestor should eventually get bus
Arbitration Options

- Bus Request
- Bus Grant
- Bus Release
- Option 1: Daisy Chain

Problems:
- Not fair
- Not fast, especially for lowest priority

Centralized and Distributed Arbitration

- Centralized:
  - Require roundtrip communication
  - (USB 1.2 only lets host initiate transfers)

- Distributed:
  - Self-selection
  - Faster
  - Require duplicated state
  - (Or cleverness and robustness…)

Where are the Busses Today?

DIMMs still sit on a bus, but typically limited to 2 DIMMs per bus.

Busses vs. Point-to-Point

- Off-chip busses are going away
  - PCI -> PCIe
  - ATA -> SATA
  - CPU interconnect -> QPI/HyperTransport

- Why?
  - Data is too fast for long wires
    - (wire = capacitor; time to reach a voltage is exponential in the size)
  - Too hard to account for multiple devices on the same wire
    - Too hard to keep wires synchronized
    - Each wire is different and wires influence each other

- Replacement:
  - SerDes (serializer/deserializer) send 64 bits one-bit-at-a-time
  - No need to keep 64 wires synchronized
  - Recover the clock as part of the signal
  - Only one wire to understand
  - But, must send data much faster (more complex, but transistors are free!)

I/O System Today

- Memory Controller on-chip
- PCIe on-chip
- Southbridge I/O interface
  - SATA
  - USB
  - Ethernet
  - Audio

Talking to I/O Devices

- Two parts:
  - How does the CPU specify the device, data, and command?
  - When does the CPU know data is ready?

- How:
  - Special Instructions: I/O Instructions
  - Special Memory Addresses: Memory Mapped

- When:
  - Ask until it’s ready: Polling
  - Alert me when it’s ready: Interrupts

- What do we need to communicate?
  - Send/receive data (data)
  - Send commands/receive status (control)
How Does the CPU Access The Device?

- What do we need to specify?
  - Device, data, and command
- Need to control (limit) access to the device
  - Security
  - Fairness
- Need to provide extensibility/flexibility
  - Different numbers of devices
  - Different interfaces for each device

How: I/O Instructions

- Format: device# and command
  - Device# indicates the device
  - E.g., keyboard input buffer is device #3
  - Command
    - Sent to the I/O bus’s data lines
    - Or on the memory bus for memory-mapped
    - From registers or memory

  - Example: x86
    - 64kB I/O space
    - IN $A, 33
    - OUT 33, $A

How: Memory Mapped

- Map portions of the address space to I/O devices
- Read and write to those addresses to access the device
- User programs prevented from accessing it through protection (TLB/pages)
- No special instructions needed
  - Just need to know where your device is “mapped” into memory
  - Protection is free through virtual memory system

Memory Mapped Idea

- How do we access the devices?
  - Simple: load/store
  - Need the addresses for operation
- What’s the problem?
  - Uses up some of our address space

(Not a problem for 64-bit machines, but is a problem for 32-bit.)

When Do We Access the Device?

- The OS needs to know when:
  - The I/O device needs attention
    - e.g., a network packet has arrived
  - The I/O device has completed an operation
    - e.g., a disk write is done
  - The I/O operation has encountered an error
    - e.g., the wireless network has gone down
- Two main ways to do this:
  - Polling:
    - The I/O device puts its status somewhere
    - The OS repeatedly checks for it to change
  - Interrupt:
    - When the I/O device status changes it grabs the processor’s attention
      by signaling an interrupt
Polling

• Example
  
  
  ```c
  while (read_status(deviceID) == 1) {
    ;
  }
  ```

• Why is this good?
  — Simple: the processor is completely in control
  — Easy to see what is happening

• Why is this bad?
  — CPU has to poll all the time to detect changes
  — Where do you put your code?

Polling Overhead

• Example:
  — 2GHz processor/100 cycles per polling operation
  — 0 cycles to switch between OS and program (!)

• Mouse: polled 30 times per second
  — Clock cycles per second for polling: 30 x 100 = 3,000
  — Fraction of processor cycles: 3/28 = 0.00015%

• Hard Disk: transfer rate 50MB/s
  — Transferred in 4-byte words
  — Need to poll (50MB/s) / 4 bytes per transfer = 12.5M times per second
  — Clock cycles for polling: 12.5M * 100 = 1.25B
  — Fraction of processor cycles: 1.25B/2B = 62.5%

Why Interrupts?

• Polling is not efficient for active devices
  — (And we ignored the cost of switching to the OS to constantly check, which is 10,000+ cycles!)


• Interrupt: “Interrupt me when you’re done and I’ll take care of it. Until then I’m going to do something else.”

• I/O interrupt is like an exception except:
  — Asynchronous (caused by an external I/O device)
  — Need more information than the cause: e.g., what does the device want?

Interrupt-Driven

• Example
  
  ```c
  while (!done) {
    calculate_next_answer();
  }
  ```

  ```c
  handle_IO_interrupt() {
    ;
  }
  ```

• Why is this good?
  — User program only halted when there is a real need

• Why is this bad?
  — Special hardware needed to cause and detect the interrupts
  — Have to save processor state
  — Multiple interrupts can happen at the same time

Interrupt Details

• Need to know which device generated the interrupt
  — Vectored interrupt (different interrupt code for every device)
  — Status register (check it when the interrupt occurs)

• Priority
  — Network and USB interrupt at the same time
  — Which one goes first? Need to prioritize

  • Handling multiple interrupts at once
    — Pressing a key when the disk is reading
    — Two techniques:
      • Masking: disable interrupts while handling others
      • Nesting: take an interrupt within an interrupt

• What happens if you disable an interrupt and it occurs?
  — Nothing — you miss it...but where does the data go?

Hardware for Interrupts
DMA: Direct Memory Access

- Wouldn’t it be better if we could just tell the device: “When you get data, store it at memory location X and tell me when you’re done.”?
- External devices transfer data directly to/from memory
- Only need an interrupt when the whole transfer is done

- But...
  - We need control circuits to keep track of where the data should go and how much data to transfer
  - Previously the CPU program did this.

(Everything fast is done this way these days because the overhead of babysitting every transaction is too high.)

Operating System Responsibilities

- Interface between I/O hardware and the program
- Three characteristics of the I/O System
  - Shared (by multiple programs)
  - Causes Interrupts (which must be handled by the OS because they access privileged resources)
  - Complex (drivers and state management are detailed and thoroughly obnoxious)
- The OS needs to hide this and provide:
  - Protection to shared resources (security)
  - Abstraction for accessing them (hide the dirty details)
  - Handle the interrupts (and respond correctly)
  - Share the resources “fairly” (whatever that means)
  - Schedule accesses for “best” performance (whatever that means: latency? throughput?)

Operating System Responsibilities

- Interface between I/O hardware and the program
- Three characteristics of the I/O System
  - Shared (by multiple programs)
  - Causes Interrupts (which must be handled by the OS because they access privileged resources)
  - Complex (drivers and state management are detailed and thoroughly obnoxious)
- The OS needs to hide this and provide:
  - Protection to shared resources (security)
  - Abstraction for accessing them (hide the dirty details)
  - Handle the interrupts (and respond correctly)
  - Share the resources “fairly” (whatever that means)
  - Schedule accesses for “best” performance (whatever that means: latency? throughput?)

OS Abstractions

- Unix: everything is a file
  - Even disks: /dev/disk0
  - USB: /dev/usb
- Same commands to access:
  - fopen Get permission to access the device
  - fread Get data from the device
  - fwrite Send data to the device
  - fseek Go to a specific location (address?)

Ethernet

- Serial “shared medium” interface
- Data divided into packets

- Preamble
- Address
- Source Address
- Type
- Body
- CRC
- Reassemble

How do we know the data wasn’t corrupted?

Device 1

Device 2

Device 3

CRC

Verifies the packet

Ethernet: Collisions

- What happens if two devices transmit at the same time? (No central arbiter...)
  - Detect the collision (read the data we transmit to see if it is corrupted by another packet)
  - Continue for 51.2µs (so everyone sees the collision and ignores the data)
  - Wait for a random (and increasing) period of time and start again
- Works really well

Summary

- I/O Performance is tricky
  - Sometimes you want latency, sometimes throughput
- Busses
  - Mastering and Arbitration
  - Split transaction vs. Address+Data lines
  - Being replaced with point-to-point due to electrical characteristics
- Accessing Devices:
  - How: Memory-Mapped vs. I/O Instructions
  - When: Polling vs. Interrupt vs. DMA
- OS
  - Must provide abstraction, security, and performance
- And... these slides are already out of date!