Direct Memory Access (DMA)

- Interrupts remove overhead of polling...
- But still requires OS to transfer data one word at a time
  - OK for low bandwidth I/O devices: mice, microphones, etc.
  - Bad for high bandwidth I/O devices: disks, monitors, etc.

Direct Memory Access (DMA)
- Transfer data between I/O and memory without processor control
- Transfers entire blocks (e.g., pages, video frames) at a time
  - Can use bus “burst” transfer mode if available
- Only interrupts processor when done (or if error occurs)

DMA Controllers

- To do DMA, I/O device attached to DMA controller
  - Multiple devices can be connected to one controller
  - Controller itself seen as a memory mapped I/O device
  - Processor initializes start memory address, transfer size, etc.
  - DMA controller takes care of bus arbitration and transfer details
  - So that’s why buses support arbitration and multiple masters!

I/O Processors

- A DMA controller is a very simple component
  - May be as simple as a FSM with some local memory
- Some I/O requires complicated sequences of transfers
  - I/O processor: heavier DMA controller that executes instructions
    - Can be programmed to do complex transfers
    - E.g., programmable network card

DMA Overhead

- Parameters
  - 500 MHz CPU
  - Interrupt handler takes 400 cycles
  - Data transfer takes 100 cycles
  - 4 MB/s, 16 B interface disk transfers data 50% of time
  - DMA setup takes 1600 cycles, transfer 1 16KB page at a time

- Processor overhead for interrupt-driven I/O?
  - 0.5 * (4M B/s)/(16 B/xfer)*[(500 c/xfer)/(500M c/s)] = 12.5%
- Processor overhead with DMA?
  - Processor only gets involved once per page, not once per 16 B
  - 0.5 * (4M B/s)/(16K B/page) * [(2000 c/page)/(500M c/s)] = 0.05%
DMA and Memory Hierarchy

- [material in class]

DMA and Address Translation

- Which addresses does processor specify to DMA controller?
- [material in class]

DMA and Caching

- Caches are good
  - Reduce CPU's observed instruction and data access latency
  + But also, reduce CPU's use of memory...
  + ...leaving majority of memory/bus bandwidth for DMA I/O

- But they also introduce a coherence problem for DMA
  - Input problem
    - DMA write into memory version of cached location
    - Cached version now stale
  - Output problem: write-back caches only
    - DMA read from memory version of "dirty" cached location
    - Output stale value

DMA and Address Translation

- Solutions to Coherence Problem
  - Route all DMA I/O accesses to cache
    + Solves problem
      - Expensive: CPU must contend for access to caches with DMA
  - Disallow caching of I/O data
    + Also works
      - Expensive in a different way: CPU access to those regions slow
  - Selective flushing/invalidations of cached data
    - Flush all dirty blocks in "I/O region"
    - Invalidate blocks in "I/O region" as DMA writes those addresses
    - The high performance solution
      - Hardware cache coherence mechanisms for doing this
      - Expensive in yet a third way: must implement this mechanism
Hardware Cache Coherence (see ECE 259)

- D$ and L2 "snoop" bus traffic
  - Observe transactions
  - Check if written addresses are resident
  - Self-invalidate those blocks
  + Doesn't require access to data part
  - Does require access to tag part
    - May need 2nd copy of tags for this
    - That's OK, tags smaller than data
- Bus addresses are physical
  - L2 is easy (physical index/tag)
  - D$ is harder (virtual index/physical tag)
    - Reverse translation? No
    - Remember: page size vs. D$ size

Designing an I/O System for Bandwidth

- Approach
  - Find bandwidths of individual components
  - Configure components you can change...
    - To match bandwidth of bottleneck component you can’t
- Example (from P&H textbook)
  - Parameters
    - 300 MIPS CPU, 100 MB/s backplane bus
    - 50K OS insns + 100K user insns per I/O operation
    - SCSI-2 controllers (20 MB/s): each accommodates up to 7 disks
    - 5 MB/s disks with \( t_{\text{seek}} + t_{\text{rotation}} = 10 \text{ ms}, 64 \text{ KB reads} \)
  - Determine
    - What is the maximum sustainable I/O rate?
    - How many SCSI-2 controllers and disks does it require?

Designing an I/O System for Bandwidth

- First: determine I/O rates of components we can't change
  - CPU: \( \frac{300 \text{ MIPS}}{150 \text{K Insns/IO}} = 2000 \text{ IO/s} \)
  - Backplane: \( \frac{100 \text{ MB/s}}{64 \text{ K B/IO}} = 1562 \text{ IO/s} \)
  - Peak I/O rate determined by bus: \( 1562 \text{ IO/s} \)
- Second: configure remaining components to match rate
  - Disk: \( \frac{1}{(10 \text{ ms/IO} + (64 \text{K B/IO}) / (5 \text{M B/s})} = 43.9 \text{ IO/s} \)
  - How many disks?
    - \( (1562 \text{ IO/s}) / (43.9 \text{ IO/s}) = 36 \text{ disks} \)
  - How many controllers?
    - \( (43.9 \text{ IO/s}) \times (64 \text{K B/IO}) = 2.74 \text{M B/s per disk} \)
    - \( (20 \text{M B/s}) / (2.74 \text{M B/s}) = 7.2 \text{ disks per SCSI controller} \)
    - \( (36 \text{ disks}) / (7 \text{ disks/SCSI-2}) = 6 \text{ SCSI-2 controllers} \)
- Caveat: real I/O systems modeled with simulation
Summary

• Role of the OS
• Device characteristics
  • Data bandwidth
  • Disks
    • Structure and latency: seek, rotation, transfer, controller delays
• Bus characteristics
  • Processor-memory, I/O, and backplane buses
  • Width, multiplexing, clocking, switching, arbitration
• I/O control
  • I/O instructions vs. memory mapped I/O
  • Polling vs. interrupts
  • Processor controlled data transfer vs. DMA
  • Interaction of DMA with memory system