Computer Systems Architecture
CMSC 411
Unit 6 – Storage Systems

Alan Sussman

November 23, 2004

Storage systems

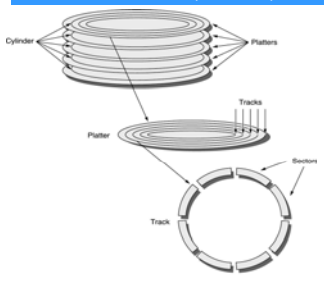
- We already know about four levels of storage:
  - registers
  - cache
  - memory
  - disk
- but we've been a little vague on how these devices are interconnected
- In this unit, we study
  - input/output units such as disks and tapes
  - buses to connect storage devices
  - I/O performance issues
  - design of file systems (won't talk much about this)

Disk and Tape Technologies

(Hard) Disks

- What it is:
  - a collection of 1-20 *platters* (like 2-sided CD's)
  - between 1 and 8 inches in diameter – 2.5 & 3.5 inch most common today
  - rotating on a central spindle
  - with 500-2500 tracks on each surface
  - divided into (maybe) 64 sectors
    - older disks: all tracks have the same number of sectors
    - current disks: outer tracks have more sectors
- larger diameter: best retrieval times
- smaller diameter: cheaper and uses less power

Disks (cont.) – Fig. 7.1



- Used for
  - file storage
  - slowest level of virtual memory during program execution

Disks (cont.)

- How information is retrieved:
  - Wait for previous requests to be filled
    Time = *queuing delay*
  - A movable arm is positioned at the correct cylinder
    Time = *seek time*
  - The system waits for the correct sector to appear under the arm
    Time = *rotational latency*
  - Then a magnetic head senses
    - the sector number
    - the information recorded in the sector
    - an error correction code

Disks (cont.)

- and the information is transferred to a buffer
  Time = *transfer time*
- The retrieval is handled by a disk controller, which may impose some extra overhead
  Time = *controller time*

- Because all of this is so expensive, might also read the next sector or two, hoping that the next information needed is located there (prefetch or *read ahead*)

Example

average seek time	5 ms
transfer rate	10MB/sec
rotation speed	8000 RPM
sector size	1024 bytes
controller overhead	.5 ms

- Average disk access time (in millisec): average seek time + average rotational delay + transfer time + controller overhead

Example (cont.)

- average seek time = 5 ms
- average rotational delay = $\frac{0.5}{8{,}000RPM} = \frac{0.5}{(8{,}000/60)RPS} = 3.75ms$
- transfer time = $\frac{1KB}{10MB/\sec} = \frac{10^3 bytes}{10^7 bytes/\sec} = 10^{-4}\sec = .1ms$
- controller overhead = .5 ms
- Total: 5 + 3.75 + .1 + .5 = 9.35 ms

Computer Systems Architecture CMSC 411 Unit 6 – Storage Systems

Alan Sussman

November 30, 2004

Administrivia

- HW #5 due today
- Project due Friday
  - questions?
- Quiz 3 scheduled for Dec. 7
  - practice quiz posted by tomorrow
- Online course evaluation available at https://www.courses.umd.edu/online_evaluation

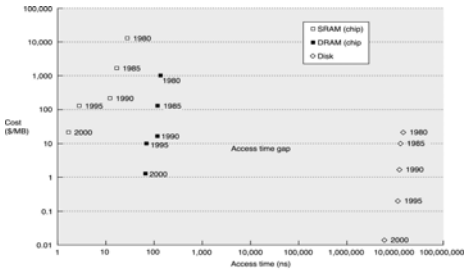
Last time

- Speculation
  - can provide precise exceptions – ROB
  - can issue multiple instructions per clock, and commit multiple per clock
  - don't speculate on expensive events (e.g. 2nd level cache misses)
  - can speculate through multiple branches
- P6 microarchitecture
  - generate RISC-like micro-operations for each IA-32 instruction
  - out-of-order, speculative pipeline with ROB

Last time (cont.)

- Storage systems
  - how a disk works
    - platters, tracks, cylinders, sectors
  - retrieval costs
    - queuing delay – wait for previous requests
    - seek time – find the right track
    - rotational latency – find the right sector
    - transfer time - read the data into a buffer (and sector number, ECC)
    - controller time – overhead in disk controller

Technology gap between memory and disk – Fig. 7.5



Competitors to disks

- *solid state disks* built from DRAMs (but needs constant power)
- optical disks: CDs and DVDs
- magnetic tapes: slower, but large capacity good for backups
- automated tape libraries: juke box technology
- flash memory – small, fast, low power

Buses

Buses

- We've seen buses before, especially in the discussion of Tomasulo's algorithm
- Main characteristic: Buses are shared by several data paths and therefore can be bottlenecks
  - CPU-memory buses: physically short, high speed, design optimized for performance
  - I/O buses: long, handle an unknown number of devices with unpredictable characteristics

Typical bus transaction

- When a READ is issued:
  - Bus begins in a wait state
  - Address sent on bus to memory, with control information to signal a read
  - When data is available, the wait signal is turned off and the data is transmitted
- When a WRITE is issued:
  - Bus begins in a wait state
  - Address sent on bus to memory, with control information to signal a write
  - Then the data is transmitted, usually with no pause

Bus options – Fig. 7.8

Option	High performance	Low cost
Bus width	separate address and data lines	multiplex address and data lines
Data width	wider is faster (e.g., 64 bits)	narrower is cheaper (e.g., 8 bits)
Transfer size	multiple words have less overhead	single-word transfer is simpler
Bus masters	multiple (need arbitration)	single (no arbitration)
Split transactions?	yes – separate request and reply gets higher bandwidth	no – continuous connection cheaper and lower latency
Clocking	synchronous	asynchronous

Who issues READs and WRITEs?

- The *bus master* does
- If the bus is between CPU and memory, then the CPU is the bus master
- If it is an I/O bus, then there might be several devices, so several bus masters, and they compete for time slices on the bus
  - In this case, buses are often *packet switched* - each device divides its message into fixed length packets, and takes turns with other devices that are transmitting

Synchronous vs. asynchronous buses

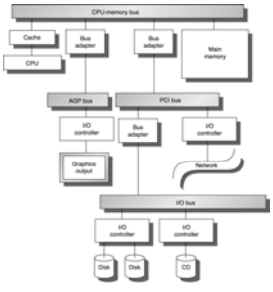
- Buses that are clocked (*synchronous*) send data and addresses at fixed times, so sender and receiver always know what to expect
  - Makes them fast and cheap
  - But restricts them to be short, because of time-lag problems
- Buses that are not clocked (*asynchronous*) use handshaking protocols to establish contact:
  - Sender puts message on bus to get the attention of receiver
  - Receiver responds
  - Sender transmits data
  - Receiver sends acknowledgement of receipt

Asynchronous buses

- Because of handshaking protocol,
  - They can be slow and expensive
  - But it allows them to be physically long and to serve a wide variety of devices
- The handshaking protocols are standardized so that device manufacturers can connect to a variety of buses

How is the I/O bus connected?

- Do we connect it to
  - the memory bus
  - or to the cache?
- Typical solution in Fig. 7.15



How does the CPU get data from the I/O bus?

- Two solutions:
  - Some (mostly older) machines have op-codes that read or write to I/O devices
  - In *memory mapped I/O*, certain physical addresses are reserved for I/O devices like disks, so those reads and writes are put on the I/O bus
- Usually I/O is *interrupt driven*, meaning that after the CPU requests a READ or WRITE, it goes on with other work until the I/O unit signals that it is finished

DMA to make this work

- To allow the CPU to proceed, need another controller to shepherd the READ or WRITE. *Direct memory access* (DMA) hardware is used to:
  - record the address and the number of bytes to be transferred
  - act as bus master, initiating each data transfer
  - interrupt the CPU when the transfer is complete
- In some cases, these controllers are really separate I/O processors

Reliability, Availability, and RAID

Failure rate vs. Availability

- *Failure rate*: concerns whether any of the hardware is broken
- *Availability*: concerns whether the system is usable, even if some pieces are broken
- Example 1: Your bank can improve the availability of the ATM system by installing two ATM machines so that one is available even if one breaks
- Example 2: Your bank can reduce the failure rate of the ATM system by installing a machine that does not break as often
  - Also increases the availability
- Generally, hope that more complicated hardware improves availability and performance, but it also may increase the failure rate

Example: Disk arrays

- Suppose a machine has an array of 20 disks
  - Case 1: If distribute the data across the disks (*striping*), then all 20 disks must be working properly in order to access the data - but throughput can be improved
  - Case 2: If store 20 copies of the data, one copy per disk, have good availability: can access the data even if some disks fail
- But reliability of the 20 disks is less than reliability of a single disk: the probability of one of the 20 disks failing is essentially 20 times the probability that a single disk will fail

Disk arrays (cont.)

- In Case 2, store multiple copies on multiple disks, called *RAID*: redundant arrays of inexpensive disks
- RAID is actually not inexpensive (because of the cost of the controllers, power supplies, and fans), so often the "I" is said to stand for "independent"
  - More than 80% of non-PC disk drive sales are now RAID, a $19B industry
  - Typically store 2 copies, not 20
  - Used when availability is critical, in applications such as:
    - airline reservations
    - medical records
    - stock market

RAID – Fig. 7.17

- There are various levels of RAID, depending on the relative importance of availability, accuracy, and cost

RAID level	# faults survived	Example data disks	Check disks	Companies
0 - Striped	0	8	0	widely used
1 - Mirrored	1	8	8	EMC, Compaq, IBM
2 - Memory-style ECC	1	8	4	
3 - Bit-interleaved parity	1	8	1	Storage Concepts
4 - Block-interleaved parity	1	8	1	Network Appliance
5 - 4 w/distributed parity	1	8	1	widely used
6 – P+Q redundancy	2	8	2	

RAID levels 0 & 1

- One copy of data: *RAID 0*
  - Data *striped* across a disk array
- Two full copies of data (*mirroring*): *RAID 1*
  - If one disk fails, go to other
  - Can also use this to distribute the load of READs
  - Most expensive RAID option
- RAID 0 and 1 can be combined
  - 1+0 (or 10) – mirror pairs of disks, then stripe across pairs
  - 0+1 (or 01) – stripe across one set of half the disks, then mirror writes to both sets

RAID 3

- Bit-interleaved parity: *RAID 3*
  - One copy of the data, stored among several disks, and one extra disk to hold a parity bit (checksum) for the others
- *Example*: Suppose have 4 data disks, and one piece of the data looks like this:

  Disk 1: 0 1 0 1 1 0 0 0
  Disk 2: 0 1 1 1 0 1 1 0
  Disk 3: 0 1 1 1 1 0 0 0
  Disk 4: 0 0 0 1 0 1 0 1
  - Then the parity bits are set by taking the sums mod 2:

    Disk 5: 0 1 0 0 0 0 1 1

RAID 3 (cont.)

- So if the data on one of the disks becomes corrupted, the parity bits on Disk 5 will be wrong, so can tell there has been a failure
  - and be able to fix it if know which disk failed
- Disadvantage: Each data access must read from all 5 disks in order to retrieve the data and check for corruption
  - also can't always tell where the error is (could even be on the parity disk)

RAID 4

- Block-interleaved parity: *RAID 4*
  - Same organization of data as RAID 3 but cheaper reads and writes
  - **Read**: Read one sector at a time, and count on the sector's own error detection mechanisms.
  - **Write**: In each write, note which bits are changing - this is enough information to change the parity bits without reading from the other disks

RAID 4 example

- If the original contents are

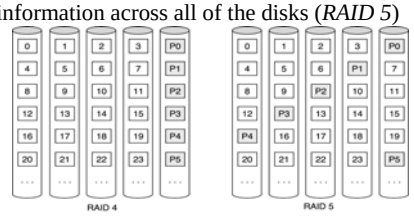
  Disk 1: 0 1 0 1 1 0 0 0
  Disk 2: 0 1 1 1 0 1 1 0
  Disk 3: 0 1 1 1 1 0 0 0
  Disk 4: 0 0 0 1 0 1 0 1
  Disk 5: 0 1 0 0 0 0 1 1
- And write

  Disk 2: 0 1 1 1 0 1 1 0 old
  Disk 2: 1 0 1 1 0 0 1 1 new
- Then since bits 0, 1, 5, and 7 changed, need to flip those parity bits:

  Disk 5: 0 1 0 0 0 0 1 1 old
  Disk 5: 1 0 0 0 0 1 1 0 new

RAID 5 – Fig. 7.19

- Disadvantage of RAID 4: Parity disk is a bottleneck, so it is better to interleave the parity information across all of the disks (*RAID 5*)



RAID 6

- Also called P+Q redundancy
  - to allow recovery from a second failure, since parity schemes only can recover from one
  - need a second extra (check) disk
  - computation is more complicated than simple parity

RAID summary

- Higher throughput than single disk
  - in either MB/sec or I/Os/sec
- Failure recovery easy
- Allows taking advantage of small size and low power requirements of small disks, and still get these advantages
  - RAIDs now dominate large-scale storage systems
- **Note**: No need to memorize the RAID levels
  - But you need to be able to explain how the example RAID levels work

I/O performance measures

I/O performance measures

- *diversity*: which I/O devices can connect to the system?
- *capacity*: how many I/O devices can connect to the system?
- *bandwidth*: throughput, or how much data can be moved per unit time
- *latency*: response time, the interval between a request and its completion
- High throughput usually means slow response time!

Throughput vs. latency

Fig. 7.24

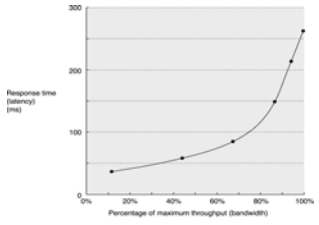


Fig. 7.25

Improving performance (cont.)

- Adding another server can decrease response time, if workload is held constant
  - but keeping work balanced between servers is difficult
- To design a responsive system, must understand what the "typical" user wants to do with it
- Each *transaction* consists of three parts:
  - entry time: the time for the user to make the request
  - system response time: the latency
  - think time: the time between system response and the next entry
- Key observation is that a faster system produces a lower think time – see Fig. 7.26

Modeling computer performance

- The usual way to model computer performance is using *queuing theory* (mathematics again)
- Unfortunately, even queuing theory does not provide a very good model, so more complicated mathematics is now being applied (e.g., stochastic differential equations)
- But, H&P only consider queuing models
  - and we don't even have time to go into that

Data Management Issues

Data Management Issues

- Two concerns we'll talk about:
  - stale data
  - DMA design
- And the book has short discussions of several more, including
  - asynchronous I/O through the OS
  - file systems – server manages blocks and maintains metadata vs. disks doing it and server uses file system protocol, such as NFS (also called NAS – network attached storage)

Stale Data

- May have copies of data in
  - cache
  - memory
  - disk
- Need to make sure that always use the most recent version
  - for use in the CPU
  - for output
- Two approaches to the problem, both having disadvantages

Stale Data (cont.)

- Approach 1: Attach the I/O bus to the cache
- Advantage: No problem of stale data, since CPU and I/O devices all see the copy in the cache
- Disadvantage:
  - All I/O data must go through the cache, even if the CPU doesn't need it, so performance is reduced
  - CPU and I/O bus must take turns accessing the cache, so arbitration hardware required

Stale Data (cont.)

- Approach 2: Attach the I/O bus to the memory

Fig. 7.15

Stale Data (cont.)

- Advantage: I/O does not slow down the CPU
- Disadvantage:
  - The I/O system may see stale data, unless we do write-through
  - CPU might see stale data if the I/O system modifies memory after the cache copied it
- Extra hardware is required to check whether I/O data is currently held in cache

DMA design

- Direct memory access hardware needs to use either
  - virtual addresses
  - or physical addresses
- Using physical addresses:
  - If the data is longer than a page, then several addresses need to be passed
  - The data may be relocated by the operating system, changing the physical address
- Virtual addresses gives a cleaner design

Designing an I/O System

Designing an I/O System

- Price, performance, and capacity issues
- Need to choose
  - which I/O devices to connect
  - how to connect them
- Example: The CPU is seldom the limiting factor for I/O performance
- Suppose the CPU can handle 10,000 I/O operations per second (IOPS)
- And suppose the average I/O size is 16 KB

I/O Systems

- The other links in the I/O chain are:
  - the *I/O controller* - suppose it adds 1 ms overhead per I/O operation
  - the *I/O bus* - suppose it is a bus that can transfer 20 MB/sec = 20 KB/ms
  - the *disk* - suppose it rotates at 7200 RPM, with 8 ms average seek time and 6 MB/sec transfer rate

I/O System Performance

- Consider the disk time first:
  - 7200 RPM = $7200/(60*10^3)$ = .12 revolutions per ms
  - 6 MB/sec = 6 KB/ms
  - So the average disk time is seek + rotational latency + transfer = 8 ms + .5 / .12 ms + 16 / 6 = 14.9 ms
- So the average time per transfer is
  - I/O controller time + bus time + disk time = 1 ms + 16 / 20 ms + 14.9 ms = 16.7 ms
- So with one controller, one bus, and one disk, can do at most
  - $1/(16.7*10^{-3})$ = 60 IOPS
- If this is not good enough, should analyze to see whether it is better to add more controllers, more buses, or more disks
- See many more examples in Section 7.11

Fallacies

- Disks never fail
  - a mean time to failure (MTTF) for one disk of 1.2M hours, or 140 years, computed by running thousands of disks for a few months, then counting the number that failed
  - but a more useful measure is the % of disks that fail in a given time period (e.g., 5 years), computed as #failed disks/total #disks
    - where # failed disks = #disks * #hours/disk /MTTF

Fallacies

- Computer systems can achieve 99.999% availability
  - that's 5 minutes per year downtime, and highly unlikely in your environment
  - in 2001, well managed servers typically available 99% to 99.9% of time
- DRAM will replace disks in desktop and server machines
  - disk manufacturers have pushed the rate of technology improvement in disks to match or exceed that of DRAM
  - instead of DRAMs killing disks, disks are killing tapes

Fallacies

- Average disk seek is for a seek of 1/3 of the cylinders
  - just a rule of thumb for seeking from one random location to another random location on a different cylinder, assuming a large number of cylinders
  - problems with that rule are that
    - seek time is not linear in distance (mechanical issues)
    - there is locality to disk accesses

Fallacies – Fig. 7.52