Lecture #26

****
Review -- 1 min
****
A Little Queuing Theory
- Little's law: L = rT
- add assumptions about arrival patterns

$$Tqueue = \frac{Tserver * (1+C) * Util\_server}{2*(1-Util\_server)}$$

- Assumptions
  - ♦ Little – weak
  - ♦ response time – stronger
- Good tool (better than, say, assuming 100% bus use)

****
Outline - 1 min
****
Connecting I/O to processor
Benchmarking

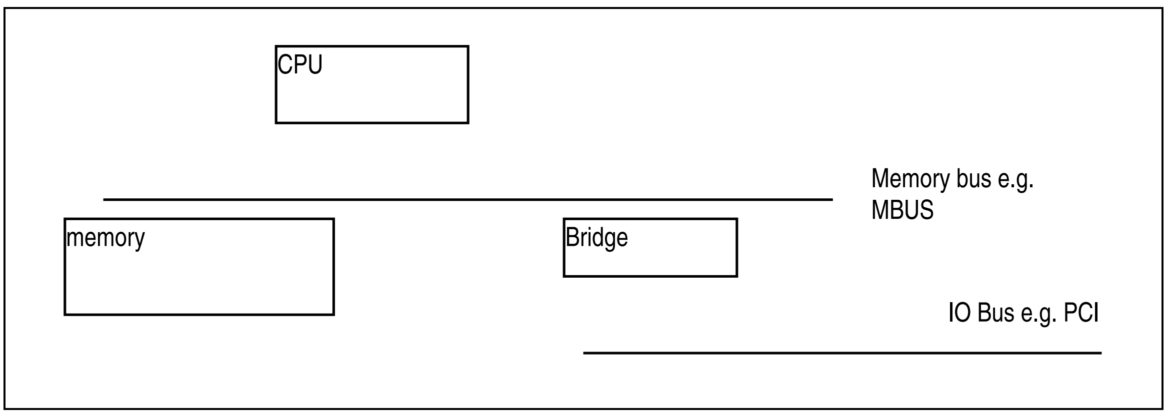
****
Preview - 1 min
****
RAID
Research in Storage Architectures
Networks

****
Lecture - 20 min
****
Processor Interface Issues
    interconnections – busses
    processor interface
    - ♦ interrupts
    - ♦ memory mapped io
    I/O Control

- polling
- interrupts
- DMA
- I/O Controllers
- I/O Processors

Interconnections – busses

Need to connection I/O to processor/memory
Common design – 2 busses
    CPU-memory
    memory bus - I/O bus

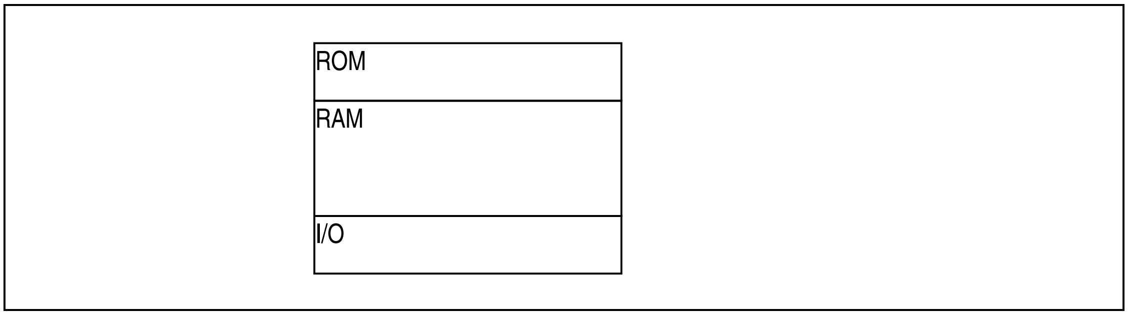


2 busses because motivated by 2 factors

CPU Bus	IO Bus
performance	cost
proprietary (improves speed)	standards-based
few devices (improves speed)	hold many devices
cheap would be nice	performance would be nice

pitfall – this approach moves I/O farther away from CPU
example – Myricom advertised Sparcstation 2 performance rather than sparcstation 10 or sparc 20. Because SS2 only had 1 bus (SBUS used for both memory and I/O), better bandwidth from network to memory.
(from memory – Sparc 2: 160 Mbit/s max, Sparc20: 136 Mbit/s max)

note interface for this approach: Memory Mapped I/O
→ CPU reads/writes "magic" addresses that correspond to I/O devices



I/O Addresses further devided – specify which I/O device, which register within device, etc…

Contrast with old approach – special I/O instructions at CPU to read and write I/O devices

Processor Interfaces

Polling v. interrupts v. DMA v. I/O processors

most basic – polling

```
loop{
    if(data available)
        read data
}
```

problem: busy wait loop not an efficient way to use CPU
(unless IO device very fast)

solutions: 1) intersperse code with I/O checks
example:
active messages network code

2) interrupts

2. interrupts
I/O device signals interrupt to CPU

how this is done – dedicated lines on bus
e.g. SBUS – 8 extra lines, 1 per SBUS device

→ CPU saves PC and jumps to interrupt service address

problem: interrupt overhead, CPU touches all data
OS + actual time to service interrupt
bad for large transfers of data

3. DMA
   device transfers data directly to/from memory
   interrupt CPU when done

limitations: works for sequential transfer
(comlplex interaction with CPU; below)

4. I/O processor
   example – linked list of I/O's to perform and
   places to put data…

Relationship to processor architecture

- Interrupts:
  stack replaced by shadow registers
  handler saves registers and re-enables higher-priority intrrupts
  interrupt types reduced in number – handler must query interrupt controller

- CPU caches cause problems for I/O
  - PIO transfers pollute cache
  - DMA ends with data not in cache → slow to use data
  - reads/writes to I/O devices cause side effects → no caching allowed
  - DMA/IO Processor is second processor → cache consistency problem
    (solutions – flush (expensive), snoop (complex)

- virtual memory makes dma comlex
- stateful processors hard to context switch
    - registers → shadow registers
    - cache → cache pollution (slowdown)

- IO Processors – hard to keep up with real CPU
  Intel spends $B's per year making processor faster
  off-loading from CPU can make sense in short term, but
  cost a lot (performance / development) in long term

- The "circle of life" – functionality tends to migrate from processor to smart I/O devices and back again (start with a nice improvement, but then keep adding functionality until IO processor is slow; in mean time CPU gets faster)

****
Admin - 3 min
****
HW4: Next Friday
Project checkpoints ---

****
Lecture - 24 min
****

Benchmarks

    "For better or worse, benchmarks shape a field"

I/O Benchmarks
    typically measure throughput
    possibly with upper limit on response time

Key issue – benchmark scaling

what if fix problem size, given 60%/year increase in DRAM capacity?

Benchmark	Size of Data	%Time in IO (1992)
IO Stones	1MB	26%
Andrew	4.5MB	4%

(1MB? Fits in L2 cache on modern machines!!!)

Observation – most benchmarks synthetic

- scaling
- hard to deal with large data sets

Self-scaling, Synthetic Benchmarks

Idea: automatically increase workload to stress system being measured

3 examples

- TPC – transaction processsing (TPC-A, TPC-B, TPC-C, TPC-D)
- NFS: SPEC SFS (aka Laddis)
- Unix I/O: Willy

Transaction Processing

TP aka OLTP (On-line transaction processing)

- Changes to a large body of shared information from many terminals, with the TP system guaranteeing proper behavior on failures
- e.g. if bank's computer fails when a customer withdraws money, the TP system would guarantee that the account is debited if the customer received the money and that the account is unchanged if the money was not received
- Airline reservation systems and banks use TP

Key idea: Atomic ***Transactions***

Each transaction ➔ 2-10 disk I/Os + 5K-20K CPU instructions per disk I/O

SW efficiency crucial to avoiding disk accesses

Classic metric: TPS (Transactions per second)

- ♦ but under what workload? How were machines configured?

TPC Benchmark history

- Early 1980's: great interest in OLTP
  - ♦ Expecting demand for high TPS(e.g. ATM machines, credit cards)
  - ♦ Each vendor picked own conditions for TPS claims, report only CPU times with widely different I/O
  - ♦ Conflicting claims → disbelief of all benchmarks → chaos in market
- 1984 – Jim Gray of Tandem distributed paper to Tandem employees and 19 other companies to propose standard benchmark
- Published "A measure of transaction processing power" Datamation, 1985 by Anonymous et. Al
  - ♦ to indicate this was an effort of a large group
  - ♦ to avoid delays of legal department of each author's firm

TP by Anon et. al

- Proposed 3 standard tests to characterize commercial OLTP
  - ♦ TP1: OLTP test "DebitCredit" – simulates ATMs
  - ♦ Batch sort
  - ♦ Batch scan
- DebitCredit
  - ♦ One type of transaction – 100 bytes each
  - ♦ recorded 3 places: account file, branch file, teller file + all events recorded in history file
- Scaling: size of account, branch, teller, history are all function of throughput

  TPS #ATMs account-file size

10	1000	0.1GB
100	10K	1.0 GB
1000	100K	10.0 GB
10,000	1000K	100.0 GB

→ each TPS → 100K account records, 10 branches, 100 ATMs

- response time: 95% transactions take < 1 second
- Configuration control: report price (initial purchase price + 5 year maintenance = cost of ownership)

Problems with TP1

Often ignored the user network to terminals

used transaction generator with no think time (made sense for vendor but not what customer would see)

Solution: hire auditor to certify results

auditors soon saw many ways to trick system

→ proposed minimum compliance list (13 pages)

still can't reproduce results

→ 1988: TPPC (Transaction processing performance council)

they create standard TPC benchmarks in 1990

New TP benchmarks

- TPC-A: Revised TP1/DebitCredit
  - Arrivals: Random (TPC) v. uniform (TP1)
  - Terminals: smart v. dumb (affects instr. Path lenght)
  - ATM scaling: 10 terminals per TPS v 100
  - branch scaling: 1 record per TPS v. 10
  - response time constraint: 90% < 2 seconds v. 95% < 1
  - full disclosure: approved by TPC
  - complete TPS/response time plot v. single point
- TPC-B: Same as TPC-A but without terminals (batch processing)
- Other efforts:

- TPC-C – complex query processing
- TPC-D – decision support

Lessons from TPC

- importance of scaling
- importance of standard
- performance/cost metric

NFS Benchmark – SPEC SFS/LADDIS

1993 – attempt by NSF companies to agree on standard benchmark

- multiple "client" load generator machines
- no caching at clients
- read and write (and stat and …) random files
- reads: 85% full block + 15% partial block
- writes: 50% full block + 15% partial block
- each client access private subdirectory (subdirectories can even be in different file systems!)
- max avg response time 50 ms
- scaling: for every 100 NFS ops/sec, increase capacity by 1GB
- Result: plot of server load (throughput) v. response time

Limitations

- low-level benchmark – NFS server specific
  - e.g. no client cache (so, an odd workload)
  - → not correspond to performance seen by client
- odd workload
  - odd access patterns (no client caches)
  - no client sharing of data
  - each client gets own subdirectory or file system

Willy – self-scaling UNIX file system benchmark

(Chen and Patterson 1993)

Self-scaling to stress different aspects of system
Examine 5 parameters

- unique bytes touched -> gives file cache size
- percentage of reads
- avg I/O request size (Bernoulli C = 1)
- percentage of sequential requests: typically 50%
- Number of processes: concurrency of workload

Idea : fix four parameters while vary 1 parameter
Search space to find high throughput

Example: Fig 6.26

Benchmark conclusions

- scaling to track technology
- TPC: price performance as normalizing config
- auditing to ensure no foul play
- throughput with restricted response time is normal measure
- practical challenges
  - ♦ large data sets → artificial workload (so can artificially generate)
  - ♦ large configurations → expensive to do tests (e.g. for high TPC need thousands of clients)

****

Summary - 1 min

****