ESE534: Computer Organization

Day 12: March 3, 2010
Defect and Fault Tolerance

Today

- Defect and Fault Tolerance
  - Problem
  - Defect Tolerance
  - Fault Tolerance

Warmup Discussion

- Where do we guard against defects and faults today?
  - [Where do we accept imperfection today?]

Motivation: Probabilities

- Given:
  - N objects
  - $P_g$ yield probability
  - What’s the probability for yield of composite system of N items? [Preclass 1]
  - Assume iid faults
  - $P(\text{N items good}) = (P_g)^N$

Probabilities

- $P_{\text{all good}}(N) = (P_g)^N$
- $P=0.999999$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$P_{\text{all good}}(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^4$</td>
<td>(0.99)</td>
</tr>
<tr>
<td>$10^5$</td>
<td>(0.90)</td>
</tr>
<tr>
<td>$10^6$</td>
<td>(0.37)</td>
</tr>
<tr>
<td>$10^7$</td>
<td>(0.000045)</td>
</tr>
</tbody>
</table>
Simple Implications

- As $N$ gets large
  - must either increase reliability
  - ...or start tolerating failures
- $N$
  - memory bits
  - disk sectors
  - wires
  - transmitted data bits
  - processors
  - transistors
  - molecules
- As devices get smaller, failure rates increase
- chemists think $P=0.95$ is good
- As devices get faster, failure rate increases

Quality Required for Perfection?

- How high must $P_g$ be to achieve 90% yield on a collection of $10^{10}$ devices?

\[
\left( P_g \right)^{10^{10}} > 0.9
\]

\[P_g > 1 \times 10^{-11}\]

Three Problems

1. **Defects**: Manufacturing imperfection
   - Occur before operation; persistent
     - Shorts, breaks, bad contact
2. **Transient Faults**: 
   - Occur during operation; transient
     - node $X$ value flips: crosstalk, ionizing particles, bad timing, tunneling, thermal noise
3. **Lifetime “wear” defects** 
   - Parts become bad during operational lifetime
     - Fatigue, electromigration, burnout...
     - ...slower
     - NBTI, Hot Carrier Injection

Defects

- Shorts example of defect
- Persistent problem
  - reliably manifests
  - Occurs before computation
- Can test for at fabrication / boot time and then avoid
  - (1st half of lecture)
Faults

- Alpha particle bit flips is an example of a fault
- Fault occurs dynamically during execution
- At any point in time, can fail
  - (produce the wrong result)
- (2nd half of lecture)

Lifetime Wear

- Starts out fine
- Over time changes
  - E.g. resistance increases until out of spec.
- Persistent
  - So can use defect techniques to avoid
- But, onset is dynamic
  - Must use fault detection techniques to recognize?

In a Nut-shell...

- Device with 10^{11} elements (100BT)
- 3 year lifetime = 10^8 seconds
- Accumulating up to 10% defects
- 10^{10} defects in 10^8 seconds
  \rightarrow 1 new defect every 10ms
- At 10GHz operation:
  - One new defect every 10^{10} cycles
  - \( P_{\text{new defect}} = 10^{-19} \)

Defect Rate

First Step to Recover

- Admit you have a problem
  (observe that there is a failure)

Detection

- How do we determine if something wrong?
  - Some things easy
    - …won’t start
  - Others tricky
    - …one and gate computes False & True \rightarrow True
- Observability
  - can see effect of problem
  - some way of telling if defect/fault present
Detection

- Coding
  - space of legal values $\ll$ space of all values
  - should only see legal
  - e.g. parity, ECC (Error Correcting Codes)
- Explicit test (defects, recurring faults)
  - ATPG = Automatic Test Pattern Generation
  - Signature/BIST = Built-In Self-Test
  - POST = Power On Self-Test
- Direct/special access
  - test ports, scan paths

Coping with defects/faults?

- **Key idea**: redundancy
- Detection:
  - Use redundancy to detect error
- Mitigating: use redundant hardware
  - Use spare elements in place of faulty elements (defects)
  - Compute multiple times so can discard faulty result (faults)
  - Exploit Law-of-Large Numbers

Defect Tolerance

Two Models

- Disk Drives (defect map)
- Memory Chips (perfect chip)

Disk Drives

- Expose defects to software
  - software model expects faults
    - Create table of good (bad) sectors
    - manages by masking out in software
    - (at the OS level)
    - Never allocate a bad sector to a task or file
  - yielded capacity varies

Memory Chips

- Provide model in **hardware** of perfect chip
- Model of perfect memory at capacity X
- Use redundancy in hardware to provide perfect model
- Yielded capacity fixed
  - discard part if not achieve
Example: Memory

- Correct memory:
  - N slots
  - each slot reliably stores last value written
- Millions, billions, etc. of bits…
  - have to get them all right?

Memory Defect Tolerance

- Idea:
  - few bits may fail
  - provide more raw bits
  - configure so yield what looks like a perfect memory of specified size

Memory Techniques

- Row Redundancy
- Column Redundancy
- Bank Redundancy

Row Redundancy

- Provide extra rows
- Mask faults by avoiding bad rows
- Trick:
  - have address decoder substitute spare rows in for faulty rows
  - use fuses to program

Column Redundancy

- Provide extra columns
- Program decoder/mux to use subset of columns
Spare Memory Column

• Provide extra columns
• Program output mux to avoid

Bank Redundancy

• Substitute out entire bank
  – e.g. memory subarray
    • include 5 banks
      – only need 4 to yield perfect
    • (N+1 sparing more typical for larger N)

Spare Bank

Yield M of N

• Preclass 4: Probability of yielding 3 of 5 things?
  – Symbolic?
  – Numerical for \( P_g = 0.9 \)?

Yield M of N

• \( P(M \text{ of } N) = P(\text{yield } N) \)
  + (N choose N-1) \( P(\text{exactly N-1}) \)
  + (N choose N-2) \( P(\text{exactly N-2}) \)...
  + (N choose N-M) \( P(\text{exactly N-M}) \)... [think binomial coefficients]

M of 5 example

• \( 1^*P^5 + 5^*P^4(1-P)^1+10P^3(1-P)^2+10P^2(1-P)^3+5P^1(1-P)^4 + 1^*(1-P)^5 \)

• Consider \( P=0.9 \)
  – \( 1^*P^5 \) 0.59  \( M=5 \ P(\text{sys})=0.59 \)
  – \( 5^*P^4(1-P)^1 \) 0.33  \( M=4 \ P(\text{sys})=0.92 \)
  – \( 10P^3(1-P)^2 \) 0.07  \( M=3 \ P(\text{sys})=0.99 \)
  – \( 10P^2(1-P)^3 \) 0.008
  – \( 5P^1(1-P)^4 \) 0.00045
  – \( 1^*(1-P)^5 \) 0.00001

Can achieve higher system yield than individual components!
**Repairable Area**

- Not all area in a RAM is repairable
  - memory bits spare-able
  - io, power, ground, control not redundant

**Consider a Crossbar**

- Allows us to connect any of N things to each other
  - E.g.
    - N processors
    - N memories
    - N/2 processors
    - + N/2 memories

**Repairable Area**

- \( P(yield) = P(\text{non-repair}) \times P(\text{repair}) \)
- \( P(\text{non-repair}) = P_{\text{Nnr}} \)
  - \( N_{\text{nr}} < \ll N_{\text{total}} \)
  - \( P > P_{\text{repair}} \)
    - e.g. use coarser feature size
  - \( P(\text{repair}) \sim P(\text{yield } M \text{ of } N) \)

**Crossbar Buses and Defects**

- Two crossbars
- Wires may fail
- Switches may fail

- Provide more wires
  - Any wire fault avoidable
    - M choose N
Crossbar Buses and Faults
• Two crossbars
• Wires may fail
• Switches may fail
• Provide more wires
  – Any wire fault avoidable
  • M choose N
  – Same idea

Simple System
• P Processors
• M Memories
• Wires

Simple System w/ Spares
• P Processors
• M Memories
• Wires
• Provide spare
  – Processors
  – Memories
  – Wires

Simple System w/ Defects
• P Processors
• M Memories
• Wires
• Provide spare
  – Processors
  – Memories
  – Wires
• ...and defects

Simple System Repaired
• P Processors
• M Memories
• Wires
• Provide spare
  – Processors
  – Memories
  – Wires
• Use crossbar to switch
together good processor and memories

In Practice
• Crossbars are inefficient [Day14-20]
• Use switching networks with
  – Locality
  – Segmentation
• …but basic idea for sparing
  is the same
Defect Tolerance Questions?

Fault Tolerance

Faults
- Bits, processors, wires
  - May fail during operation
- Basic Idea same:
  - Detect failure using redundancy
  - Correct
- Now
  - Must identify and correct online with the computation

Transient Sources
- Effects
  - Thermal noise
  - Timing
  - Ionizing particles
    - α particle $10^5$ to $10^6$ electrons
    - Calculated gates with 15–30 electrons Day 7
      - Even if CMOS restores, takes time

Voltage and Error Rate

Errors versus Frequency

[Austin et al. -- IEEE Computer, March 2004]

[Bowman, ISSCC 2008]
Scaling and Error Rates

- Increasing Error Rates

SEU/bit Norm to 130nm
- Increasing Error Rates

- Technology (nm)

- Logic

- Cache arrays

Increasing Error Rates
- 2X bitlatch count increase per generation
- Technology (nm)

Technology (nm)

SEU/bit Norm to 130nm

Source: Carter/Intel

Simple Memory Example

- **Problem**: bits may lose/change value
  - Alpha particle
  - Molecule spontaneously switches
- **Idea**:
  - Store multiple copies
  - Perform majority vote on result

Redundant Memory

- Like M-choose-N
- Only fail if > (N-1)/2 faults
- P=0.9
- P(2 of 3)
  - All good: \((0.9)^3 = 0.729\)
  - Any 2 good: \(3(0.9)^2(0.1) = 0.243\)
  - \(= 0.971\)

Better: Less Overhead

- Don’t have to keep N copies
- Block data into groups
- Add a small number of bits to detect/correct errors

Row/Column Parity

- Think of NxN bit block as array
- Compute row and column parities
  - (total of 2N bits)
Row/Column Parity

• Think of N×N bit block as array
• Compute row and column parities  
  – (total of 2N bits)
• Any single bit error

InClass Exercise

• Which Block has an error?
• What correction do we need?

In Use Today

• Conventional DRAM Memory systems  
  – Use 72b ECC (Error Correcting Code)
  – On 64b words [12.5% overhead]
  – Correct any single bit error
  – Detect multibit errors
• CD blocks are ECC coded  
  – Correct errors in storage/reading

Row/Column Parity

• Think of N×N bit block as array
• Compute row and column parities  
  – (total of 2N bits)
• Any single bit error
• By recomputing parity  
  – Know which one it is
  – Can correct it

RAID

• Redundant Array of Inexpensive Disks
• Disk drives have ECC on sectors  
  – At least enough to detect failure
• RAID-5 has one parity disk  
  – Tolerate any single disk failure
  – E.g. 8-of-9 survivability case
  – With hot spare, can rebuild data on spare
Interconnect

- Also uses checksums/ECC
  - Guard against data transmission errors
  - Environmental noise, crosstalk, trouble
    sampling data at high rates…
- Often just detect error
- Recover by requesting retransmission
  - E.g. TCP/IP (Internet Protocols)

Interconnect

- Also guards against whole path failure
- Sender expects acknowledgement
- If no acknowledgement will retransmit
- If have multiple paths
  - …and select well among them
  - Can route around any fault in interconnect

Interconnect Fault Example

- Send message
- Expect
  Acknowledgement
- If Fail
  - No ack
  - Retry
  - Preferably with different resource
Interconnect Fault Example

• If Fail → no ack
  – Retry
  – Preferably with different resource

Compute Elements

• Simplest thing we can do:
  – Compute redundantly
  – Vote on answer
  – Similar to redundant memory

Ack signals success

Compute Elements

• Unlike Memory
  – State of computation important
  – Once a processor makes an error
    • All subsequent results may be wrong
• Response
  – "reset" processors which fail vote
  – Go to spare set to replace failing processor

In Use

• NASA Space Shuttle
  – Uses set of 4 voting processors
• Boeing 777
  – Uses voting processors
    • Uses different architectures for processors
    • Uses different software
    • Avoid Common-Mode failures
    – Design errors in hardware, software

Forward Recovery

• Can take this voting idea to gate level
  – VonNeuman 1956
• Basic gate is a majority gate
  – Example 3-input voter
• Alternate stages
  – Compute
  – Voting (restoration)
• Number of technical details…
• High level bit:
  – Requires $P_{\text{fail}} > 0.996$
  – Can make whole system as reliable as individual gate

Majority Multiplexing

Maybe there's a better way…
Rollback Recovery

- Commit state of computation at key points
  - to memory (ECC, RAID protected...)
  - ...reduce to previously solved problem of protecting memory
- On faults (lifetime defects)
  - recover state from last checkpoint
  - like going to last backup....
  - ...(snapshot)

Defect vs. Fault Tolerance

- Defect
  - Can tolerate large defect rates (10%)
    - Use virtually all good components
    - Small overhead beyond faulty components
- Fault
  - Require lower fault rate (e.g. VN <0.4%)
    - Overhead to do so can be quite large

Summary

- Possible to engineer practical, reliable systems from
  - Imperfect fabrication processes (defects)
  - Unreliable elements (faults)
- We do it today for large scale systems
  - Memories (DRAMs, Hard Disks, CDs)
  - Internet
  - ...and critical systems
  - Space ships, Airplanes
- Engineering Questions
  - Where invest area/effort?
    - Higher yielding components? Tolerating faulty components?
    - Where do we invoke law of large numbers?
    - Above/below the device level

Admin

- HW5 due Friday
- Reading for Monday 3/15 on web

Big Ideas

- Left to itself:
  - reliability of system << reliability of parts
- Can design
  - system reliability >> reliability of parts [defects]
  - system reliability ~= reliability of parts [faults]
- For large systems
  - must engineer reliability of system
  - ...all systems becoming "large"
Big Ideas

• Detect failures
  – static: directed test
  – dynamic: use redundancy to guard
• Repair with Redundancy
• Model
  – establish and provide model of correctness
    • Perfect component model (memory model)
    • Defect map model (disk drive model)