CS184b: Computer Architecture (Abstractions and Optimizations)

Day 21: May 30, 2003
Defect and Fault Tolerance

Today

- Defect and Fault Tolerance
  - Problem
  - Defect Tolerance
  - Fault Tolerance
- EAS Questionnaire??? (10 min)

Probabilities

- Given:
  - N objects
  - P yield probability
- What's the probability for yield of composite system of N items?
  - Asssume iid faults
  - P(N items good) = P^N

Probabilities

- P(N items good) = P^N
- $N=10^6$, P=0.999999
- P(all good) ~= 0.37
- $N=10^7$, P=0.999999
- P(all good) ~= 0.000045

Simple Implications

- As N gets large
  - must either increase reliability
  - …or start tolerating failures
- N
  - memory bits
  - disk sectors
  - wires
  - transmitted data bits
  - processors

Increase Reliability?

- $P_{sys} = P^N$
- Want: P_{sys} = constant
- $c = \ln(P_{sys}) = N \ln(P)$
- $\ln(P) = \ln(P_{sys})/N$
- P=Nth root of P_{sys}

Problems

Two “problems”

- Shorts, breaks
  - wire/node X shorted to power, ground, another node
- Noise
  - node X value flips
    - crosstalk
    - alpha particle
    - bad timing

Defects

- Shorts example of defect
- Persistent problem
  - reliably manifests
- Occurs before computation
- Can test for at fabrication / boot time and then avoid

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)

First Step to Recover

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

Detection

- Determine if something wrong?
  - Some things easy
    - ….won't start
  - Others tricky
    - …one and gate computes F*T=>T
- Observability
  - can see effect of problem
  - some way of telling if defect/fault present

Detection

- Coding
  - space of legal values < space of all values
  - should only see legal
  - *e.g.* parity, redundancy, ECC
- 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)

Defect Tolerance

Two Models

- Disk Drives
- Memory Chips

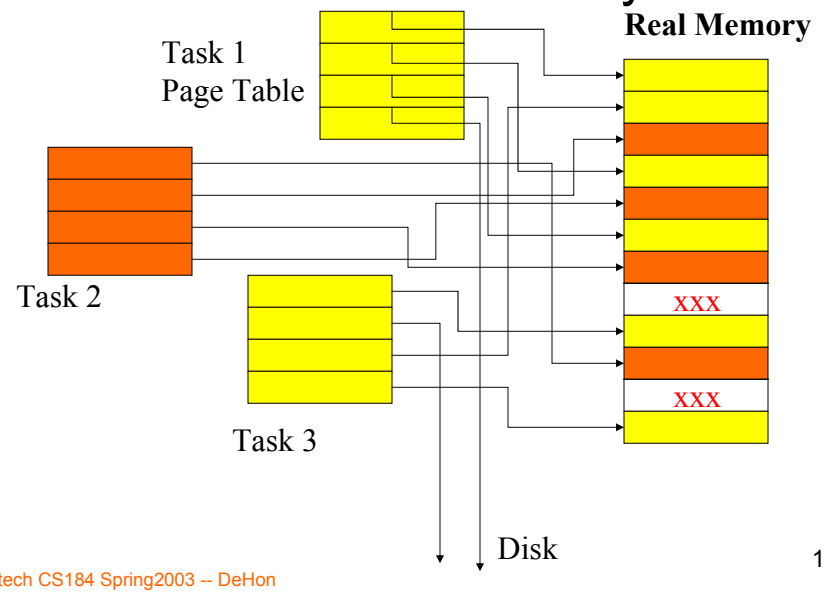
Disk Drives

- Expose faults to software
  - software model expects faults
    - Create table of good (bad) sectors
  - manages by masking out in software
    - (at the OS level)
  - yielded capacity varies

VM/Disk Drive Model

- On startup OS checks physical memory
  - Notes bad pages
  - Treat all bad pages as permanently allocated to the “bad page” process
- Never map user (OS) to a “bad page”
  - User never sees physical addresses
  - Doesn’t expect contiguous physical memory

Page Tables and Defective Memory



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

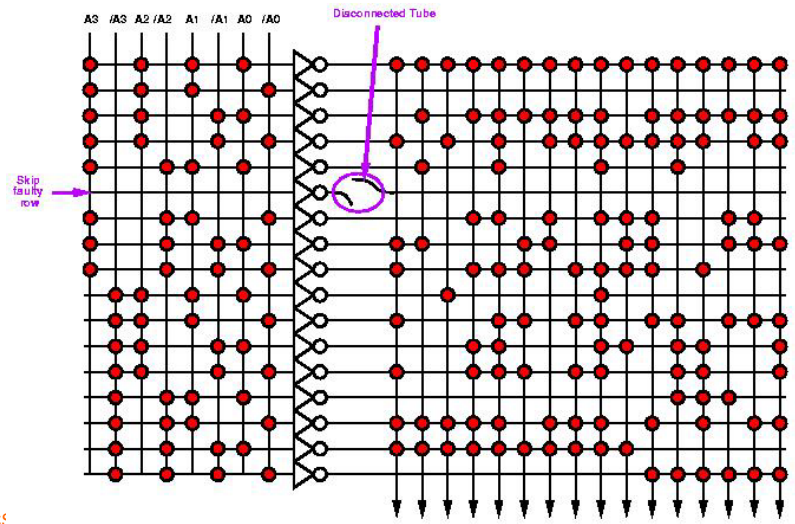
Memory Techniques

- Row Redundancy
- Column Redundancy
- Block Redundancy

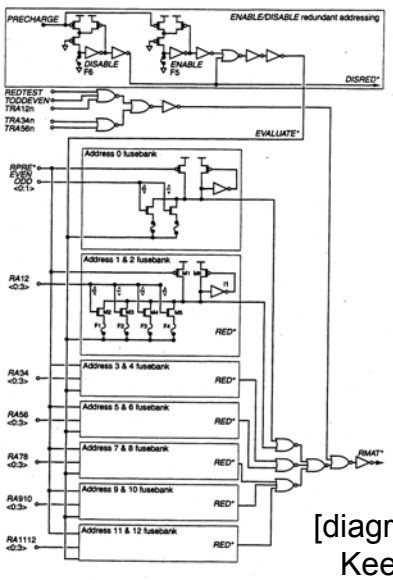
Row Redundancy

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

Spare Row



Row Redundancy



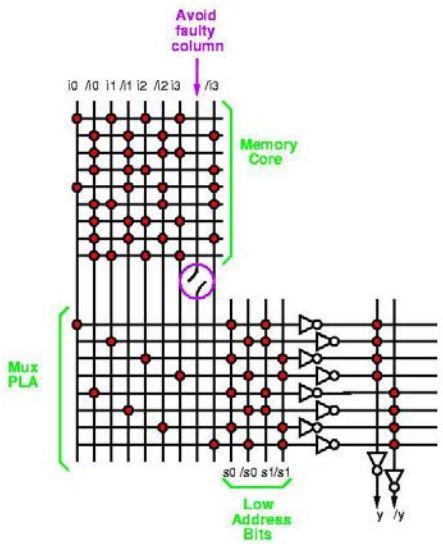
[diagram from Keeth&Baker 2001]

Column Redundancy

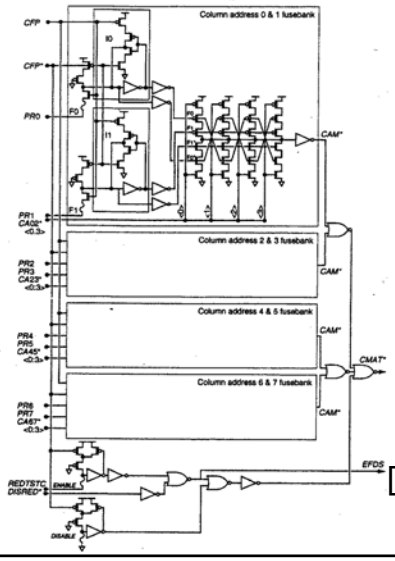
- Provide extra columns
- Program decoder/mux to use subset of columns

Spare Memory Column

- Provide extra columns
- Program output mux to avoid



Column Redundancy

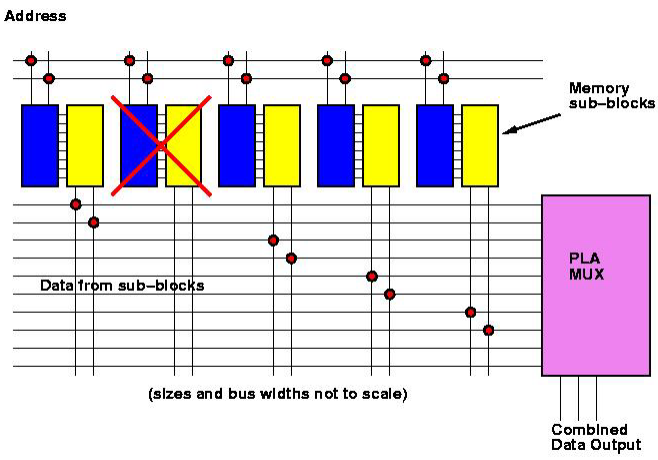


[diagram from Keeth&Baker 2001]

Block Redundancy

- Substitute out entire block
  - *e.g.* memory subarray
    - include 5 blocks
      - only need 4 to yield perfect
    - (N+1 sparing more typical for larger N)

Spare Block



Yield M of N

- P(M of N) = P(yield N)
  + (N choose N-1) P(exactly N-1)
  + (N choose N-2) P(exactly N-2)…
  + (N choose N-M) P(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(sys)=0.59
  - $5*P^4(1-P)^1$ 0.33 M=4 P(sys)=0.92
  - $10P^3(1-P)^2$ 0.07 M=3 P(sys)=0.99
  - $10P^2(1-P)^3$ 0.008
  - $5P^1(1-P)^4$ 0.00045
  - $1*(1-P)^5$ 0.00001

Repairable Area

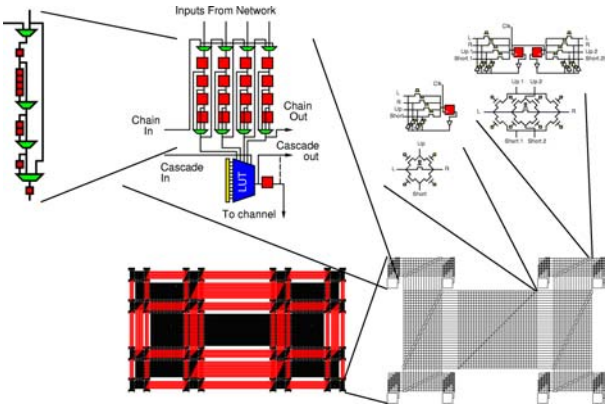
- Not all area in a RAM is repairable
  - memory bits spare-able
  - io, power, ground, control not redundant

Repairable Area

- P(yield) = P(non-repair) * P(repair)
- P(non-repair) = P^N
  - N<<Ntotal
  - Maybe P > Prepair
    - e.g. use coarser feature size
- P(repair) ~ P(yield M of N)

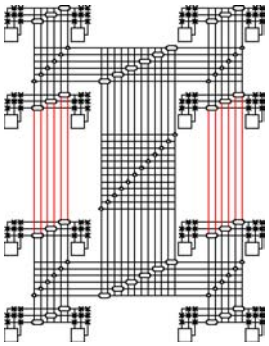
Consider HSRA

- Contains
  - wires
  - luts
  - switches



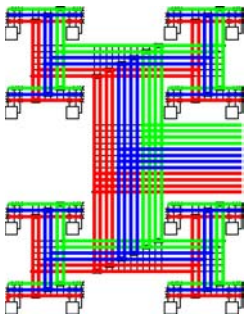
HSRA

- Spare wires
  - most area in wires and switches
  - most wires interchangeable
- Simple model
  - just fix wires



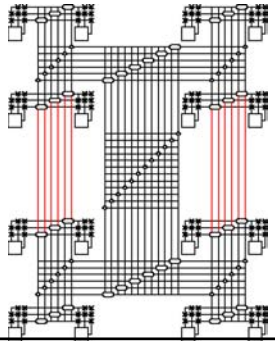
HSRA "domain" model

- Like "memory" model
- spare entire domains by remapping
- still looks like perfect device



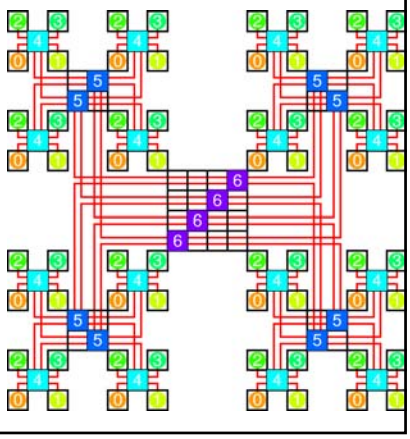
HSRA direct model

- Like “disk drive” model
- Route design around known faults
  - designs become device specific



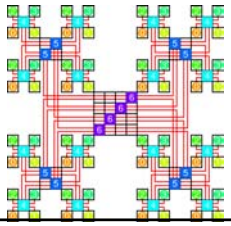
HSRA: LUT Sparing

- All LUTs are equivalent
- In pure-tree HSRA
  - placement irrelevant
  - *i.e.* which endpoint as long as match logical tree
- skip faulty LUTs



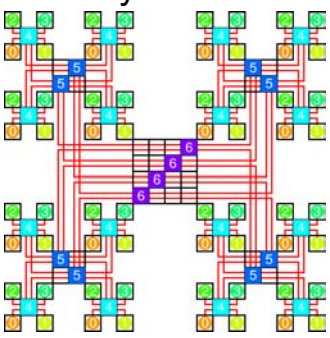
Simple LUT Sparing

- Promise N-1 LUTs in subtree of some size
  - *e.g.* 63 in 64-LUT subtree
  - shift try to avoid fault LUT
  - tolerate any one fault in each subtree



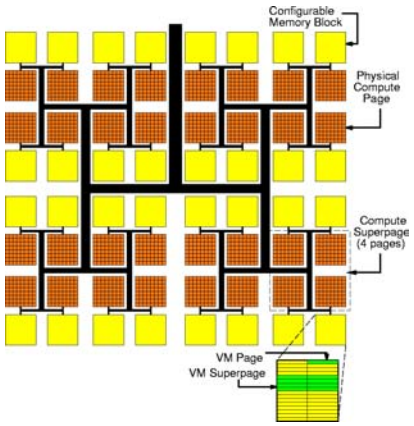
More general LUT sparing

- "Disk Drive" Model
- Promise M LUTs in N-LUT subtree
  - do unique placement around faulty LUTs



SCORE Array

- Has memory and HSRA LUT arrays

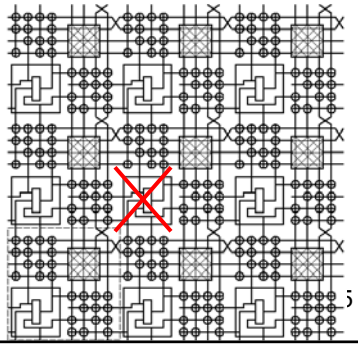


SCORE Array

- …but already know how to spare
  - LUTs
  - interconnect
    - in LUT array
    - among LUT arrays and memory blocks
  - memory blocks
- Example how can spare everything in universal computing block

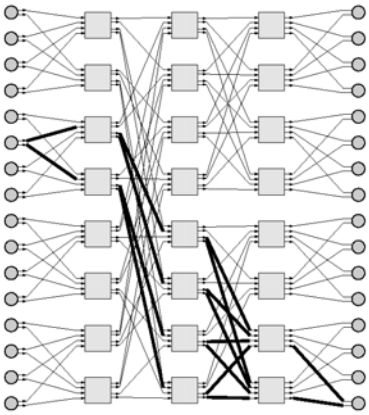
…but

- Important not to expose adjacency
  - *E.g.* physically contiguous real memory
  - *E.g.* Physical adjacency in mesh
- How spare mesh?



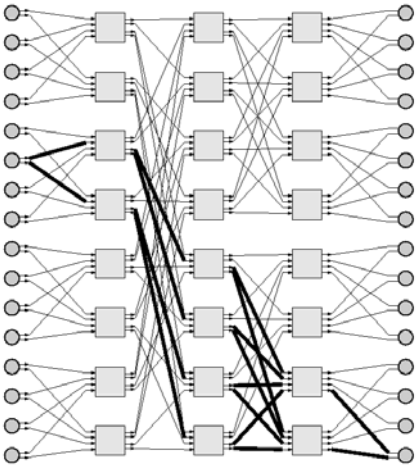
Transit Multipath

- Butterfly (or Fat-Tree) networks with multiple paths
  - showed last time



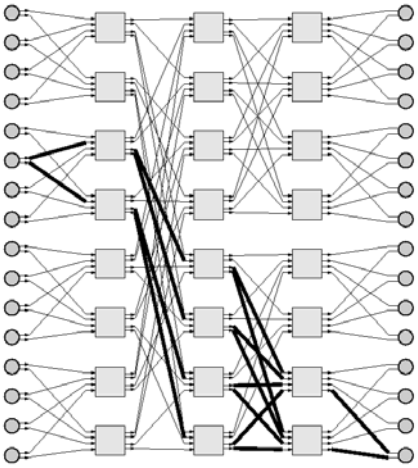
Multiple Paths

- Provide bandwidth
- Minimize congestion
- Provide **redundancy** to tolerate faults



Routers May be faulty (links may be faulty)

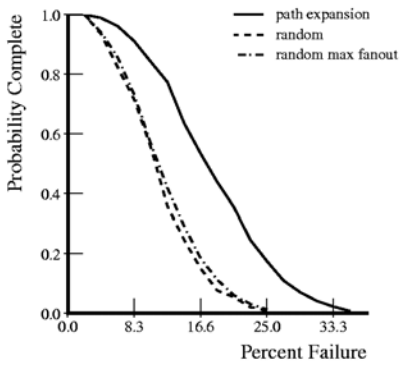
- Static
  - always corrupt message
  - not route or misroute message
- Dynamic
  - occasionally corrupt or misroute



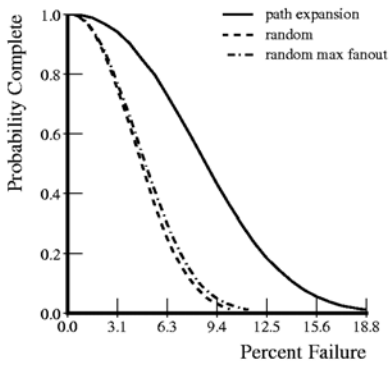
Metro: Static Faults

- Turn off
  - faulty ports (make look busy)
  - ports connected to faulty channels
  - ports connected to faulty routers
- As long as paths remain between all communication endpoints
  - still functions

Multibutterfly Yield

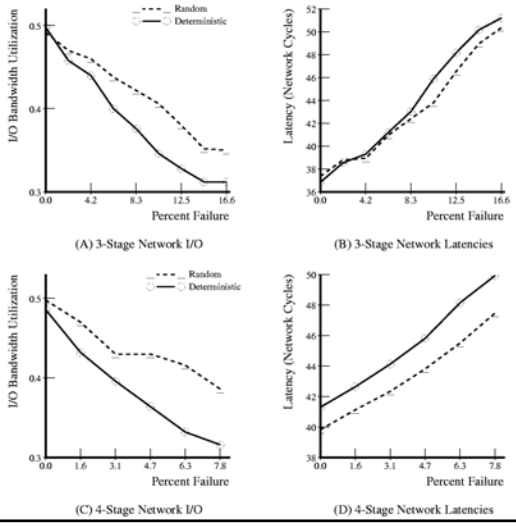


(A) 3-Stage Network Completeness



(B) 4-Stage Network Completeness

Multibutterfly Performance w/ Faults



Fault Tolerance

Metro: dynamic faults

- Detection: Check success
  - checksums on packets to see data intact
  - check destination (arrived at right place)
  - acknowledgement from receiver
    - know someone received correctly
- If fail
  - resend message
    - same as blocked route case

Metro: dynamic faults

- Consequence
  - may have faulty components
  - as long as
    - detection strong
    - there is a non-faulty path
  - will eventually deliver an intact message
    - may deliver multiple times if fault in ack
    - hence earlier concern about idempotence

Memory: Dynamic Faults

- Error Correcting Codes (ECC)
- Provide enough redundancy to
  - detect most any errors
  - correct typical errors
- Simple scheme:
  - row and column parity
    - 2 sqrt(N) bits – correct any single bit error
- …better schemes in practice
  - [Caltech has whole course on this…EE127]

Processing Faults?

- Simplest model detection:
  - parallel checking
    - run N copies in parallel
    - compare results

Processor/Gate Fault Handling

- What do on fault?
  - Stop (not do anything wrong)
    - maybe just restart
      - adequate if soft error
    - maybe "reconfigure" to substitute out faulty processor
  - Vote
    - if have enough redundancy take majority
    - Von Neumann shows:
      - Can contain error rate of system to error rate of single component

Checkpoint and Rollback

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

Together

- Examples of handling faults in
  - processing
  - storage
  - interconnect
- All components of our system

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

Big Ideas

- Detect failures
  - static: directed test
  - dynamic: use redundancy to guard
- Repair with Redundancy
- Model
  - establish and provide model of correctness
    - perfect model part (memory model)
    - visible defects in model (disk drive model)