Reliable Architectures

Joel Emer
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

http://www.csg.csail.mit.edu/6.823
Strike Changes State of a Single Bit
Impact of Neutron Strike on a Si Device

- Secondary source of upsets: alpha particles from packaging

Strikes release electron & hole pairs that can be absorbed by source & drain to alter the state of the device.
Cosmic Rays Come From Deep Space

- Neutron flux is higher at higher altitudes
  - 3x - 5x increase in Denver at 5,000 feet
  - 100x increase in airplanes at 30,000+ feet
Physical Solutions are hard

• Shielding?
  – No practical absorbent (e.g., approximately > 10 ft of concrete)
  – This is unlike Alpha particles which are easily blocked

• Technology solution: SOI?
  – Partially-depleted SOI of some help, effect on logic unclear
  – Fully-depleted SOI may help, but is challenging to manufacture

• Circuit level solution?
  – Radiation hardened circuits can provide 10x improvement with significant penalty in performance, area, cost
  – 2-4x improvement may be possible with less penalty
Triple Modular Redundancy (Von Neumann, 1956)

V does a majority vote on the results
Dual Modular Redundancy (eg., Binac, Stratus)

- Processing stops on mismatch
- Error signal used to decide which processor be used to restore state to other
Pair and Spare Lockstep (e.g., Tandem, 1975)

- Primary creates periodic checkpoints
- Backup restarts from checkpoint on mismatch
Redundant Multithreading (e.g., Reinhardt, Mukherjee, 2000)

- Writes are checked
Component Protection

- Fujitsu SPARC in 130 nm technology (ISSCC 2003)
  - 80% of 200k latches protected with parity
  - versus very few latches protected in commodity microprocessors
Strike on a bit (e.g., in register file)

- **Bit Read?**
  - yes
  - no

  - **Bit has error protection?**
    - yes
      - benign fault
      - no error
    - no
      - detection & correction
      - no error

  - **affects program outcome?**
    - yes
      - SDC
    - no
      - benign fault
      - no error

  - **affects program outcome?**
    - yes
      - True DUE
    - no
      - False DUE

**SDC = Silent Data Corruption, DUE = Detected Unrecoverable Error**
Metrics

• Interval-based
  - MTTF = Mean Time to Failure
  - MTTR = Mean Time to Repair
  - MTBF = Mean Time Between Failures = MTTF + MTTR
  - Availability = MTTF / MTBF

• Rate-based
  - FIT = Failure in Time = 1 failure in a billion hours
  - 1 year MTTF = $10^9 / (24 \times 365)$ FIT = 114,155 FIT
  - SER FIT = SDC FIT + DUE FIT

Hypothetical Example

  Cache: 0 FIT
  + IQ: 100K FIT
  + FU: 58K FIT

  Total of 158K FIT
Cosmic Ray Strikes: Evidence & Reaction

• Publicly disclosed incidence


  – Sun Microsystems found cosmic ray strikes on L2 cache with defective error protection caused Sun’s flagship servers to crash, R. Baumann, IRPS Tutorial on SER, 2000.

# Vulnerable Bits Growing with Moore’s Law

Typical SDC goal: 1000 year MTBF
Typical DUE goal: 10-25 year MTBF
Architectural Vulnerability Factor (AVF)

$$AVF_{bit} = \text{Probability Bit Matters}$$

$$= \frac{\# \text{ of Visible Errors}}{\# \text{ of Bit Flips from Particle Strikes}}$$

$$FIT_{bit} = \text{intrinsic FIT}_{bit} \times AVF_{bit}$$
Architectural Vulnerability Factor
Does a bit matter?

• Branch Predictor
  – Doesn’t matter at all (AVF = 0%)  

• Program Counter
  – Almost always matters (AVF ~ 100%)
Statistical Fault Injection (SFI) with RTL

+ Naturally characterizes all logical structures

- RTL not available until late in the design cycle
- Numerous experiments to flip all bits
- Generally done at the chip level
  - Limited structural insight
Architecturally Correct Execution (ACE)

- ACE path requires only a subset of values to flow correctly through the program’s data flow graph (and the machine)
- Anything else (un-ACE path) can be derated away
Example of un-ACE instruction: Dynamically Dead Instruction

Most bits of an un-ACE instruction do not affect program output
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

\[
\text{Average number of ACE bits in a cycle} = \frac{\text{Total number of bits in the structure}}{4}
\]
Little’s Law for ACEs

\[ N_{ace} = T_{ace} \times L_{ace} \]

\[ AVF = \frac{N_{ace}}{N_{total}} \]
Computing AVF

- **Approach is conservative**
  - Assume every bit is ACE unless proven otherwise

- **Data Analysis using a Performance Model**
  - Prove that data held in a structure is un-ACE

- **Timing Analysis using a Performance Model**
  - Tracks the time this data spent in the structure
Dynamic Instruction Breakdown

- DYNAMICALLY DEAD: 20%
- PERFORMANCE INST: 1%
- PREDICATED FALSE: 7%
- NOP: 26%
- ACE: 46%

Average across Spec2K slices
Mapping ACE & un-ACE Instructions to the Instruction Queue

Architectural un-ACE

Micro-architectural un-ACE
ACE Lifetime Analysis (1)
(e.g., write-through data cache)

• Idle is unACE

<table>
<thead>
<tr>
<th>Fill</th>
<th>Read</th>
<th>Read</th>
<th>Evict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Valid</td>
<td>Valid</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td>Idle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Assuming all time intervals are equal
• For 3/5 of the lifetime the bit is valid
• Gives a measure of the structure’s utilization
  – Number of useful bits
  – Amount of time useful bits are resident in structure
  – Valid for a particular trace
ACE Lifetime Analysis (2) (e.g., write-through data cache)

- Valid is not necessarily ACE

- ACE % = AVF = 2/5 = 40%

- Example Lifetime Components
  - ACE: fill-to-read, read-to-read
  - unACE: idle, read-to-evict, write-to-evict
ACE Lifetime Analysis (3)
(e.g., write-through data cache)

- Data ACEness is a function of instruction ACEness

- Second Read is by an unACE instruction

- AVF = 1/5 = 20%
Instruction Queue

ACE percentage = AVF = 29%
Strike on a bit (e.g., in register file)

- **Bit Read?**
  - yes
    - Bit has error protection?
      - yes
        - detection & correction
          - no error
      - no
        - detection only
          - no error
    - no
      - benign fault
        - no error

- Affects program outcome?
  - yes
    - SDC
  - no
    - benign fault
      - no error

**SDC = Silent Data Corruption, DUE = Detected Unrecoverable Error**
DUE AVF of Instruction Queue with Parity

- True DUE AVF: 29%
- False DUE AVF: 33%
- Uncommitted: 6%
- Neutral: 16%
- Dynamically Dead: 11%
- Idle & Misc: 38%

CPU2000
Asim
Simpoint
Itanium®2-like
Coping with Wrong-Path Instructions (assume parity-protected instruction queue)

• Problem: not enough information at issue
The $\pi$ (Possibly Incorrect) Bit (assume parity-protected instruction queue)

At commit point, declare error only if not wrong-path instruction and $\pi$ bit is set
Sources of False DUE in an Instruction Queue

• Instructions with uncommitted results
  – e.g., wrong-path, predicated-false
  – solution: $\pi$ (possibly incorrect) bit till commit

• Instruction types neutral to errors
  – e.g., no-ops, prefetches, branch predict hints
  – solution: anti-$\pi$ bit

• Dynamically dead instructions
  – instructions whose results will not be used in future
  – solution: $\pi$ bit beyond commit
Thank you!