Reliable Architectures

Joel Emer
Computer Science & Artificial Intelligence Lab
M.I.T.
Event Changes State of a Single Bit
Event Changes State of a Single Bit

- Soft Error – Changes that are not permanent
- Hard Error – Changes that are permanent
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.

Transistor Device
Cosmic Rays Come From Deep Space

- Neutron flux is higher at higher altitudes
  - 3–5x increase in Denver at 5,000 feet
  - 100x increase in airplanes at 30,000+ feet
Basics of Charge Generation

Cosmic rays of >1 GeV result in neutrons of >1 MeV

<table>
<thead>
<tr>
<th>Energy (eV)</th>
<th>Electron-Hole Pairs</th>
<th>Charge (Femtocoulombs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6 eV</td>
<td>1</td>
<td>3.2 x 10^{-4}</td>
</tr>
<tr>
<td>1 MeV</td>
<td>~2.8 x 10^5</td>
<td>~44</td>
</tr>
<tr>
<td>1 GeV</td>
<td>~2.8 x 10^8</td>
<td>~44 x 10^3</td>
</tr>
</tbody>
</table>

In 2010:
- Critical charge on a DRAM: ~25 fCoulomb
- Critical charge on an SRAM: <4 fCoulomb
Cosmic Ray Strikes: Evidence & Reaction

- Publicly disclosed incidences
  
  - 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.
  
  
  - In 2003, a "single-event upset" was blamed for an electronic voting error in Schaerbeekm, Belgium. A bit flip in the electronic voting machine added 4,096 extra votes to one candidate.
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?**
  - Partially-depleted SOI of some help, effect on logic unclear
  - Fully-depleted SOI may help, but is challenging to manufacture
  - FinFETs are showing significantly lower vulnerability

- **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
V does a majority vote on the results
Dual Modular Redundancy
(e.g., BINAC 1949, Stratus)

- Processing stops on mismatch
- Error signal used to decide which processor be used to restore state to other

Diagram:

- Mismatch?
- Error?
- M
- C
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

Leading Thread

Trailing Thread

Fault?
Component Protection

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

- Bit Read?
  - yes
    - Bit has error protection?
      - no
        - Benign fault
          - no error
          - no error
        - detection &
          - correction
          - no error
      - yes
        - Affects program
          - outcome?
            - yes
              - SDC
            - no
              - Benign fault
                - no error
    - yes
      - 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 = 109 / (24 * 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
Number of Vulnerable Bits Growing with Moore’s Law

![Graph showing the number of vulnerable bits over years with a 12x GAP and typical SDC and DUE goals.]

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

$AVF_{\text{bit}} = \text{Probability Bit Matters}$
Architectural Vulnerability Factor (AVF)

$$AVF_{\text{bit}} = \frac{\text{# of Visible Errors}}{\text{# of Bit Flips from Particle Strikes}}$$
Architectural Vulnerability Factor (AVF)

\[ AVF_{\text{bit}} = \text{Probability Bit Matters} \]

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

\[ \text{FIT}_{\text{bit}} = \text{intrinsic FIT}_{\text{bit}} \times AVF_{\text{bit}} \]
Statistical Fault Injection (SFI) with RTL
Statistical Fault Injection (SFI) with RTL

Simulate strike on latch
Statistical Fault Injection (SFI) with RTL

Simulate strike on latch

Logic

0

output
Statistical Fault Injection (SFI) with RTL

- Simulate strike on latch
- Check whether fault propagates to architectural state
Statistical Fault Injection (SFI) with RTL

1. Simulate strike on latch
2. Check whether fault propagates to architectural state
3. Naturally characterizes all logical structures
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

simulate strike on latch

check whether fault propagates to architectural state
Architectural Vulnerability Factor

Does a bit matter?

- Branch Predictor

- Program Counter
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

\[ \text{AVF} = \text{fraction of cycles a bit contains ACE state} \]
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

T = 1

ACE% = 2/4
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

T = 2
ACE% = 1/4
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

T = 3

ACE% = 0/4
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

T = 4

ACE% = 3/4
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

\[
\frac{(2 + 1 + 0 + 3)}{4}
\]
Vulnerability of a structure

AVF = fraction of cycles a bit contains ACE state

\[
\text{AVF} = \frac{(2 + 1 + 0 + 3)}{4}
\]

= 4

= Average number of ACE bits in a cycle

= Total number of bits in the structure
Little’s Law for ACEs

\[ \overline{N}_{ace} = \overline{T}_{ace} \times \overline{L}_{ace} \]

\[ AVF = \frac{\overline{N}_{ace}}{N_{total}} \]
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
ACE Lifetime Analysis (1)
(e.g., write-through data cache)

• Idle is unACE

• 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%
Dynamic Instruction Breakdown

Average across Spec2K slices

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

MIT 6.5900 Fall 2022
April 22, 2019
L19-27
Mapping ACE & un-ACE Instructions to the Instruction Queue

- NOP
- Prefetch
- ACE Inst
- ACE Inst
- Wrong-Path Inst
- Idle

Architectural un-ACE

Micro-architectural un-ACE
Mapping ACE & un-ACE Instructions to the Instruction Queue

- NOP
- Prefetch
- ACE Inst
- Ex-ACE Inst
- Wrong-Path Inst
- Idle

Architectural un-ACE

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

- Bit Read?
  - yes
  - no
    - Benign fault no error
  - Bit has error protection?
    - yes
      - detection & correction
      - no error
    - no
      - detection only
      - 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
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)
Coping with Wrong-Path Instructions (assume parity-protected instruction queue)
Coping with Wrong-Path Instructions
(assume parity-protected instruction queue)
Coping with Wrong-Path Instructions
(assume parity-protected instruction queue)
Coping with Wrong-Path Instructions (assume parity-protected instruction queue)
Coping with Wrong-Path Instructions
(assume parity-protected instruction queue)
Coping with Wrong-Path Instructions (assume parity-protected instruction queue)

DECLARE ERROR ON ISSUE
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)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit (assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit (assume parity-protected instruction queue)

POST ERROR IN $\pi$ BIT ON ISSUE
The $\pi$ (Possibly Incorrect) Bit (assume parity-protected instruction queue)

![Diagram](image_url)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
The $\pi$ (Possibly Incorrect) Bit
(assume parity-protected instruction queue)
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!

Next Lecture: Transactional Memory