Reliable Architectures

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

http://www.csg.csail.mit.edu/6.823
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

Transistor Device

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

<table>
<thead>
<tr>
<th>Energy (eV)</th>
<th>Electron-Hole Pairs</th>
<th>Charge (Femtocoulombs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6ev</td>
<td>1</td>
<td>3.2x10^-4</td>
</tr>
<tr>
<td>1MeV</td>
<td>~2.8x10^5</td>
<td>~44</td>
</tr>
<tr>
<td>1Gev</td>
<td>~2.8x10^8</td>
<td>~44x10^3</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 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.
  
  
  - In 2010 it was suggested that the sudden acceleration in Toyata cars might be due to soft errors, “Cosmic rays offered as acceleration cause,” Freepress.com, 2010.
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
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
Strike on a bit (e.g., in register file)

- Bit Read?
  - yes
  - Bit has error protection?
    - yes
      - benign fault
      - no error
    - no
      - detection & correction
        - no error
      - detection only
    - no
      - 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
  - $MTTF = \text{Mean Time to Failure}$
  - $MTTR = \text{Mean Time to Repair}$
  - $MTBF = \text{Mean Time Between Failures} = MTTF + MTTR$
  - Availability = $\frac{MTTF}{MTBF}$

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

Hypothetical Example

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

  Total of 158K FIT
Typical SDC goal: 1000 year MTBF
Typical DUE goal: 10-25 year MTBF
Architectural Vulnerability Factor (AVF)

$\text{AVF}_{\text{bit}} = \text{Probability Bit Matters}$

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

$\text{FIT}_{\text{bit}} = \text{intrinsic FIT}_{\text{bit}} \times \text{AVF}_{\text{bit}}$
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

\[
\text{Average number of ACE bits in a cycle} = \frac{\sum \text{ACE bits per cycle}}{\text{Total number of bits in the structure}}
\]

\[
\frac{2 + 1 + 0 + 3}{4} = \frac{6}{4} = \frac{3}{2}
\]
Little’s Law for ACEs

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

\[ AVF = \frac{\bar{N}_{ace}}{\bar{N}_{total}} \]
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%
- NOP: 26%
- ACE: 46%
- PREDICATED FALSE: 7%

Average across Spec2K slices
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 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%
Instruction Queue

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

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

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

CPU2000
Asim
Simpoint
Itanium®2-like

March 16, 2016
http://www.csg.csail.mit.edu/6.823
Sanchez & Emer
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!