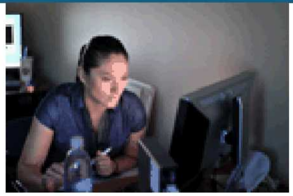
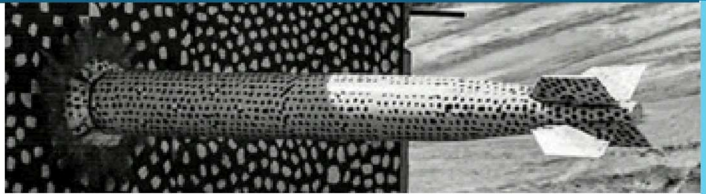
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-10029C

Fault Tracking and Modeling in Advanced Node Processors of Single Event Effects





Presented By

Matthew Cannon

Collaborators

Arun Rodrigues, Dolores Black, Jeff Black, Luis Bustamante, Ben Feinberg, Heather Quinn
Lawrence Clark, John Brunhaver, Hugh Barnaby, Michael McLain, Sapan Agarwal
and Matthew Marinella

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Introduction

Advanced systems today contain many different components and technologies

- CPUs, GPUs, FPGAs, memories, etc.

Simulations help us analyze performance and power tradeoffs at a system level

But simulations do not necessarily help us analyze SEE tradeoffs

- How do SEEs in one system module affect another?
- How do SEEs propagate throughout the system?
- Do all my modules need the same level of protection/mitigation?

SEE Testing on Processors and Heterogeneous Systems

Two options:

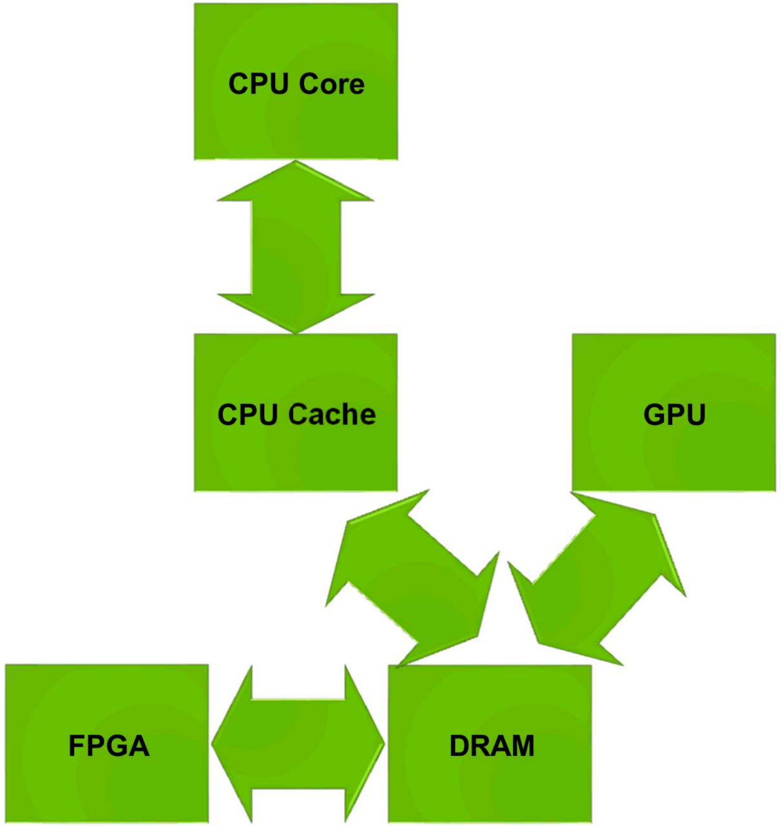
- Fault inject accessible registers over debug port
- Radiation testing (run benchmark, compare to golden)

SEE testing on complex systems has a limited scope

- Sometimes only done at block level
  - CPU, GPU and FPGA have x,y,z cross-sections
- Failure analysis limited
  - E.g., CPU failed, system took wrong action, system hang

Limited access to internal state of the system

- Which pipeline stage failed in the CPU?
- Who wrote the bad data to memory?



Proposed Solution

Leverage simulator models of components/modules for reliability testing

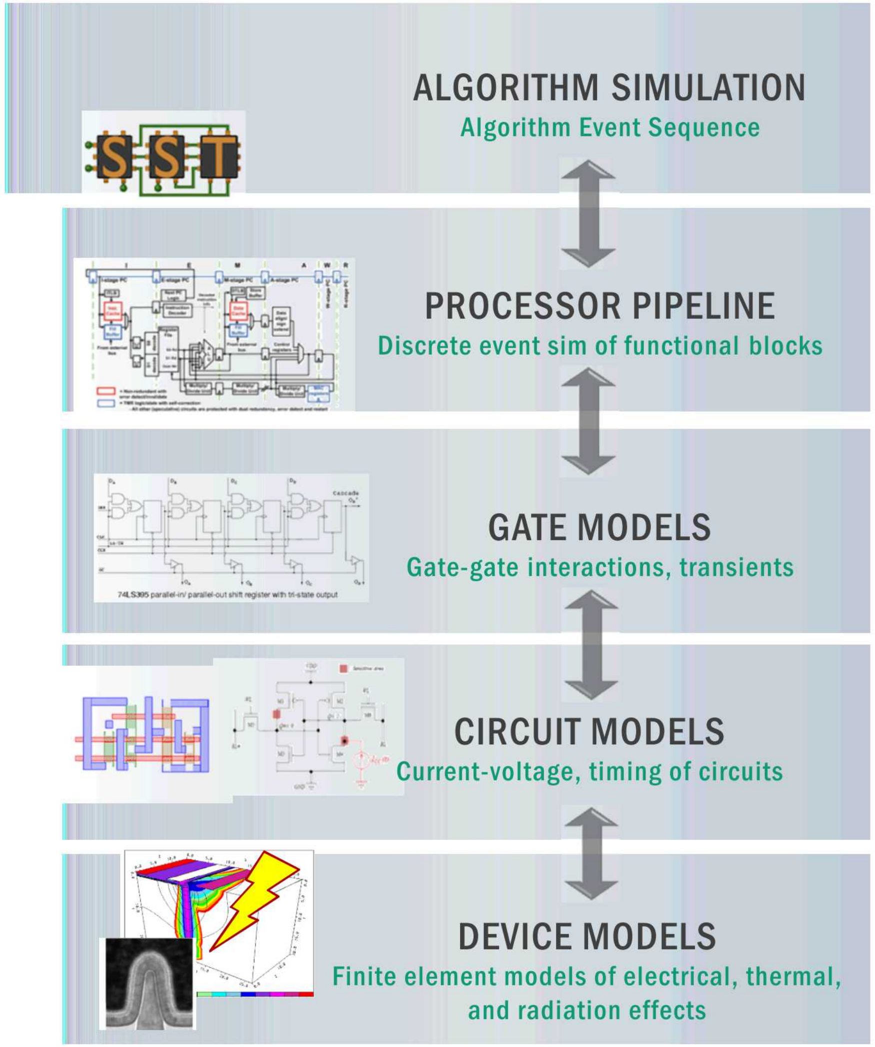
- Match registers/logic in hardware to architectural models
- Use new tools to convert synthesized netlists into C simulator code (future work)

Fault injection rates determined by:

- Target technology (e.g. 14nm Fin-Fet)
- Voltage and circuit timing
- Logic masking (from the gates)

Improvement over previous work

- No hand-waving of SEU rates
- Full pipeline model of the processor (all registers are injectable)
- Fault tracking built into register data structrures



ALGORITHM SIMULATION

Algorithm Event Sequence

PROCESSOR PIPELINE

Discrete event sim of functional blocks

GATE MODELS

Gate-gate interactions, transients

CIRCUIT MODELS

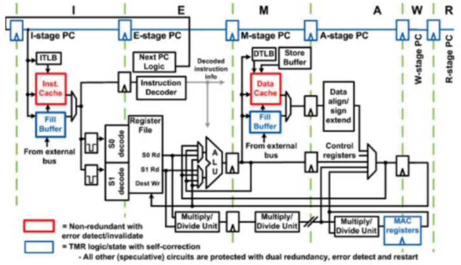
Current-voltage, timing of circuits

DEVICE MODELS

Finite element models of electrical, thermal, and radiation effects

Fault Injection Algorithm and Technology Investigations

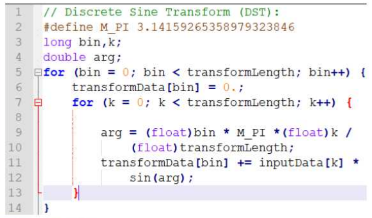
Discrete Event Simulation of Processor Pipeline
Structural Simulation Toolkit – SST



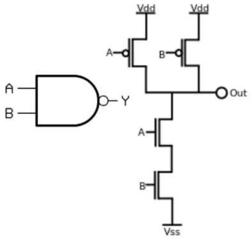
CPU Instructions, Assembly

Pipeline stage error model

C/C++ code

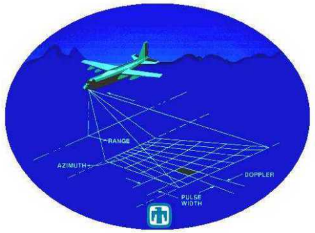


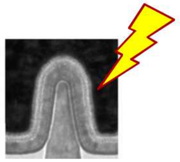
```
// Discrete Sine Transform (DST):
#define M_PI 3.14159265358979323846
long bin,k;
double arg;
for (bin = 0; bin < transformLength; bin++) {
    transformData[bin] = 0.;
    for (k = 0; k < transformLength; k++) {

        arg = (float)bin * M_PI *(float)k /
            (float)transformLength;
        transformData[bin] += inputData[k] *
            sin(arg);
    }
}
```



Gate level faults, bit flipping

Algorithm





Transistor level radiation models

Structural Simulation Toolkit

High performance simulator used to model highly concurrent systems

Model entire heterogenous system at varying levels of fidelity

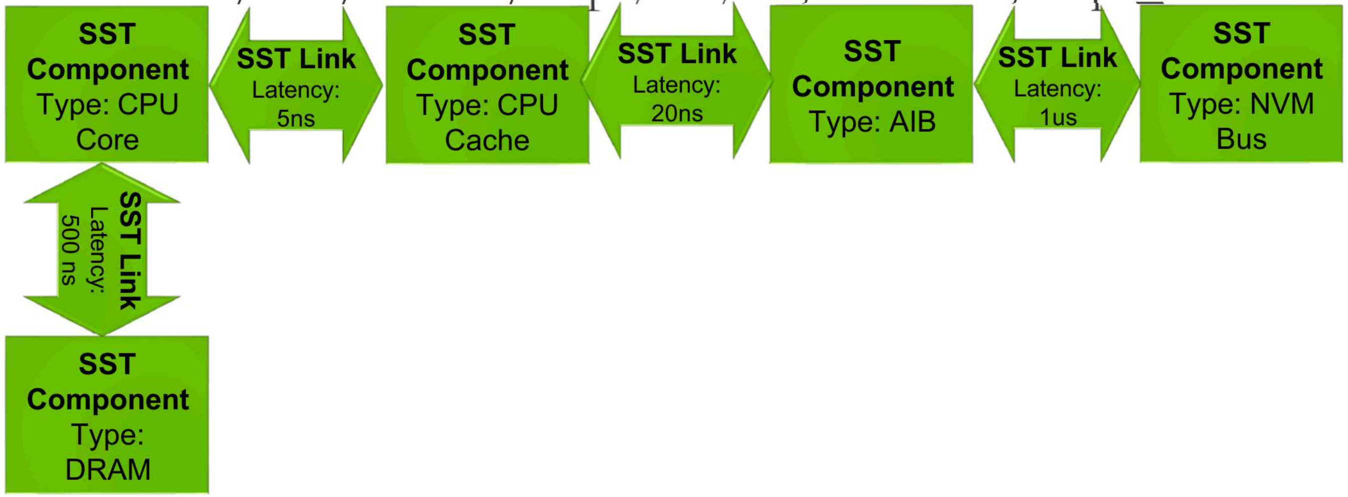
- Models hardware and algorithms running on that hardware

Model computational result, timing and energy

- Extended to inject and track faults

http://sst-simulator.org/

https://github.com/afrodri/sst-elements/tree/afrodri/mips/src/sst/elements/mips_4kc



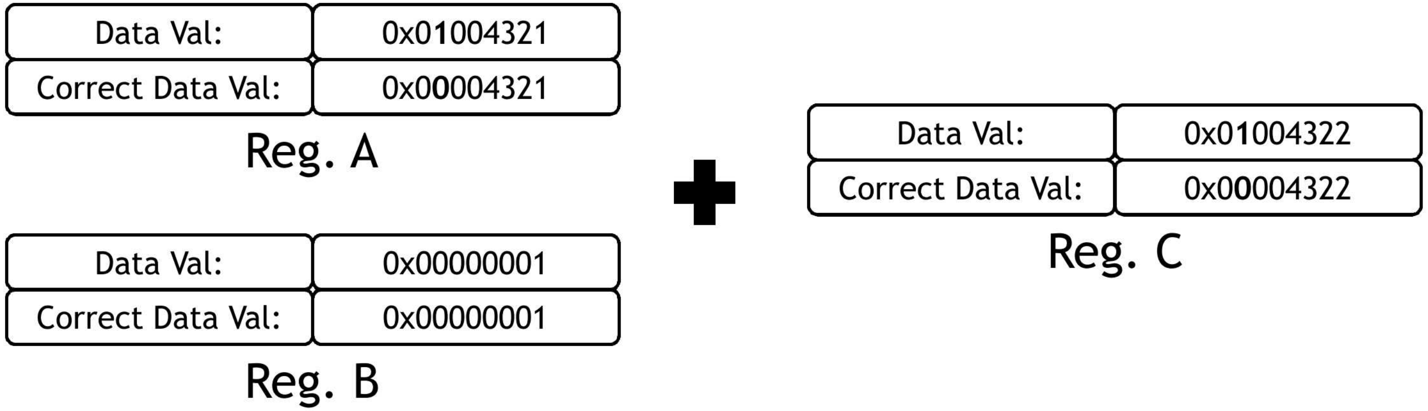
Fault Tracking within SST

Modify register data structure to track two values:

- Current value (possibly faulty value flowing through simulator due to fault injection)
- Correct value (fault-free value)

Used to determine how a fault spreads through the system

- Can determine when it is quashed (and how)
- Can determine how far it spreads through the system
- Can determine failure trace



Data Val:	0x01004321
Correct Data Val:	0x00004321

Reg. A

Data Val:	0x00000001
Correct Data Val:	0x00000001

Reg. B

Data Val:	0x01004322
Correct Data Val:	0x00004322

Reg. C

Fault Injection Capabilities

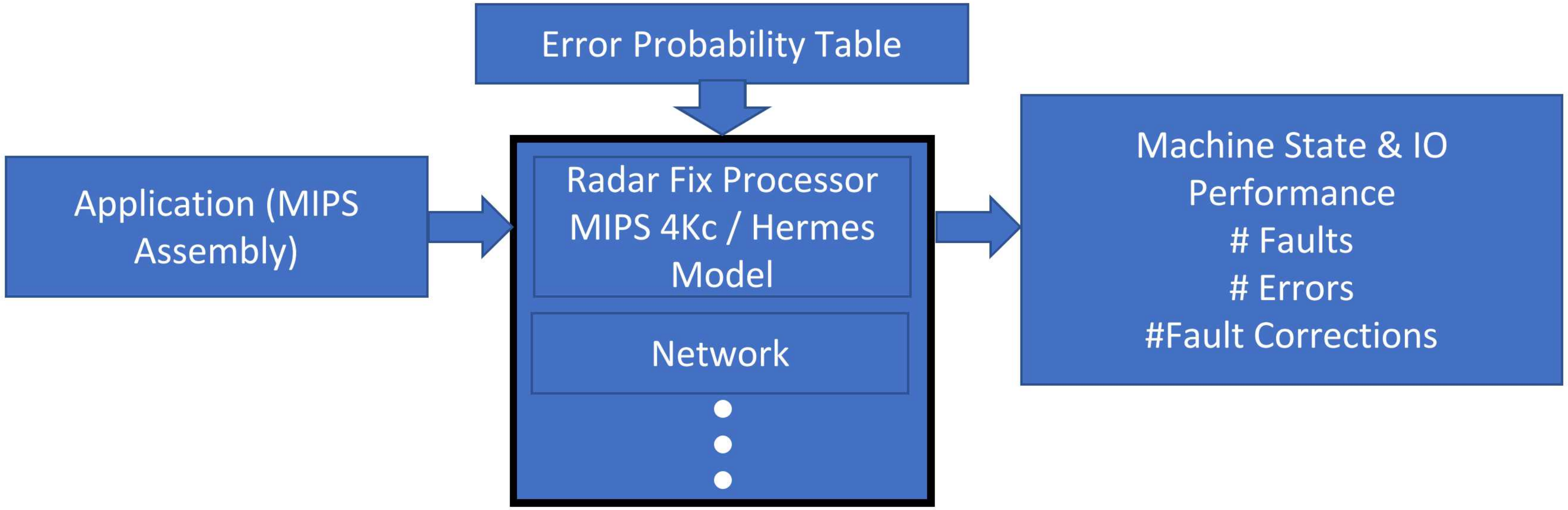
Can inject faults at the beginning of every clock cycle

- Randomly, or
- Precomputed table (for repeatability)

Error probability table can be adjusted for environment/technology

- Probabilities calculated from logical masking, register size, etc.

Allows for targeted or system wide fault injection



Error Types

Silent Data Corruption (SDC)

- Program completed, but results were corrupted

Terminated

- Program failed to complete, usually due to an illegal memory operation

Timeout

- Program still running after 4x times number of (normal) cycles have completed, probably stuck in an infinite loop

Correct

- Program completed and results are correct

Case Study : HERMES Processor

Radiation hardened by design

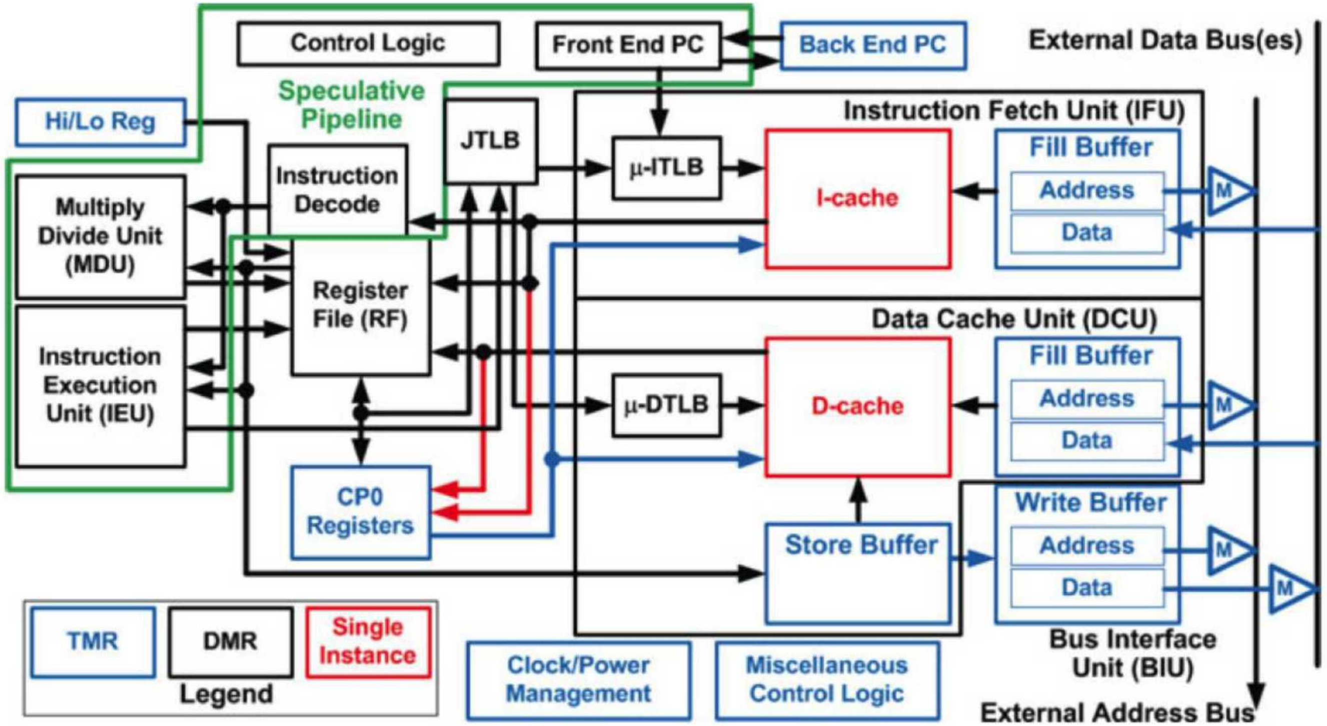
Faster and lower energy then triple redundancy, reports errors to algorithm

Caches are dual redundant and invalidated on an error

Logic that can be re-run if incorrect is protected by dual redundancy (DMR)

Correction is software controlled

Critical logic is protected by triple redundancy (TMR)

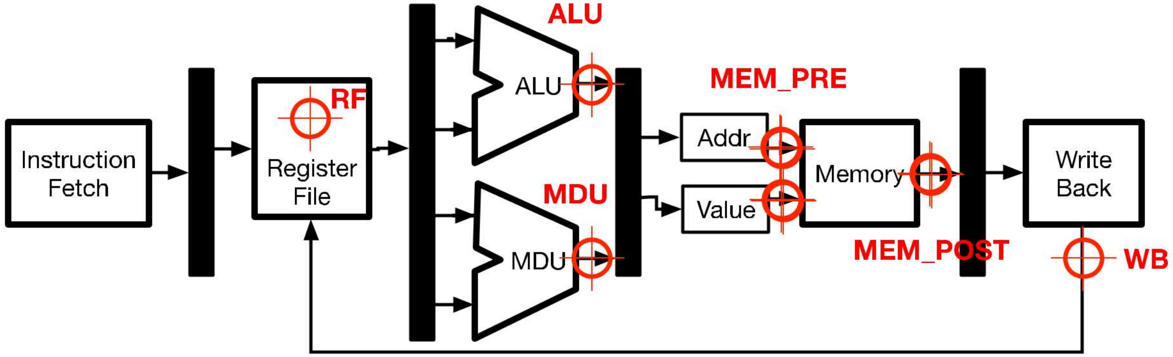


Register Modeling

Clspim MIPS pipeline model was integrated into SST.

Registers found in synthesized netlist were found & mapped into the pipeline model

- RF : Corrupt random register
- ALU/MDU : Corrupt registers used for calculations or output
- MEM_PRE : Corrupt address or store value
- MEM_POST : Corrupt value read from memory
- WB : Corrupt value written back to register file



Software Benchmarks

Matrix Multiply (12x12 w/ 32-bit unsigned integers)

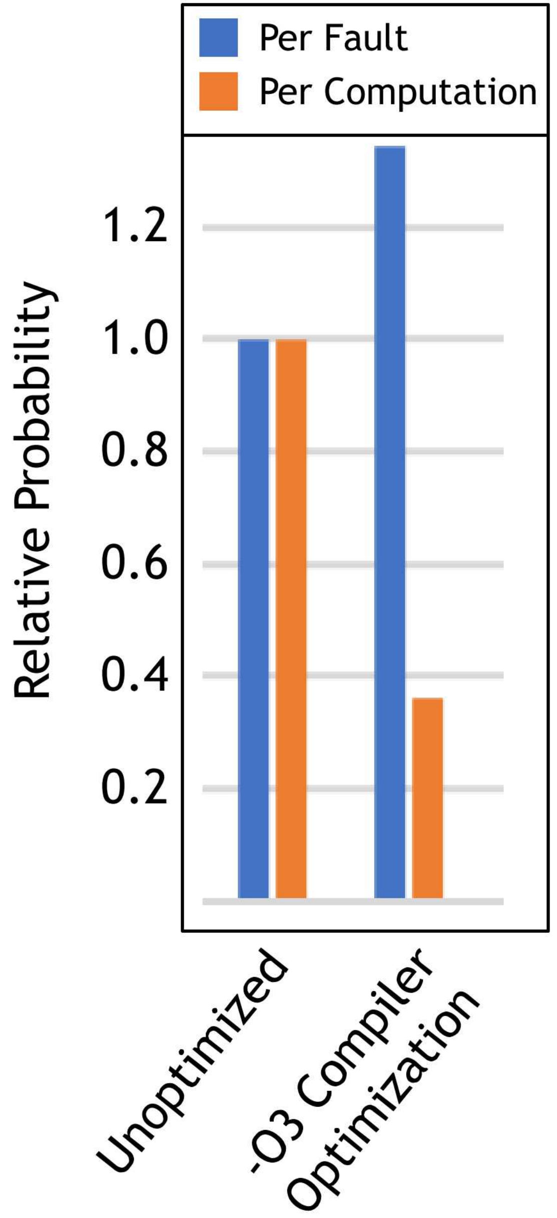
- Uses triple-nested loop

Variations of MM used

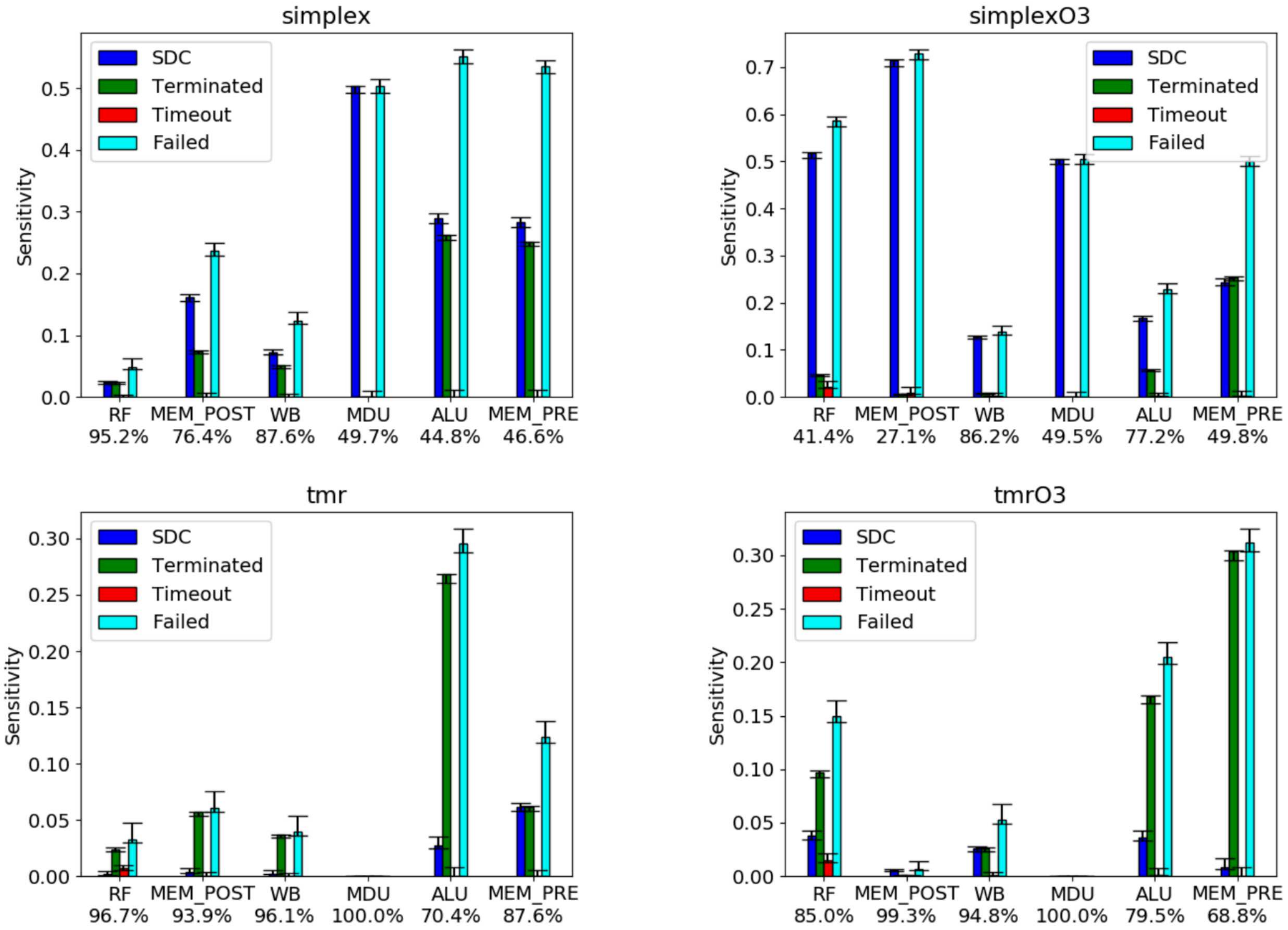
- Compiler optimizations (O2)
- Software redundancy (DMR, TMR)

Initial results demonstrate expected results

- Optimizations make each instruction more vulnerable
- But optimizations make program less vulnerable (fewer instructions/faster execution)



Fault Injection Results



Future Work

Current focus is on creating our error probability tables

- Using MRED to determine rad characteristics of our targeted technology
- Analyzing netlist to determine natural logical masking effects

Create simulation model directly from synthesized netlist

- Simulation models already exist for post-synthesis debug – extend to allow fault tracking capabilities

Extend fault injection studies

- Perform in-depth studies of timing effects of faults

Add more software to benchmark suite

- AES, qsort, etc.

Perform radiation test on HERMES processor and compare results

Conclusion

Current SEE studies on complex systems lack insight into system state that causes failure

- Limited resources to gather real-time, system state information

High performance simulators can be used to perform SEE studies on complex systems

- Ability to track faults within the system
- Easier failure-cause analysis

HERMES processor can be used to validate simulated, fault-injection approach

- HERMES can dump internal state upon detected SEU

Initial fault studies produce expected results

- Will be used to perform more complex fault tracking studies