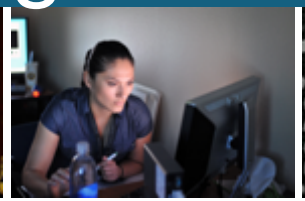
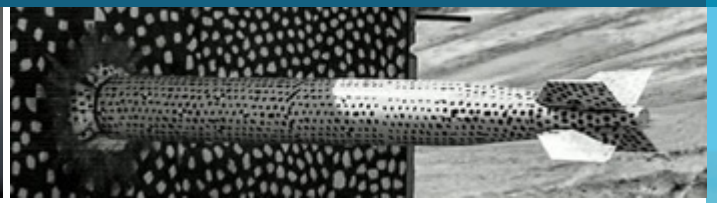
SAND2020-12163C

Sandia National Laboratories

Multiscale System Modeling of Single Event Induced Faults in Advanced Node Processors





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

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

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

SEEs are a key radiation susceptibility for modern, highly scaled process technologies

Understanding SEEs are difficult for microprocessors

- Number of SEE locations
- Inability to observe the architecture fully
- Susceptible to SDC, crashes and halts from SEUs and SETs

System designers have a need to understand how errors flow through microprocessors

Fault modeling for large-scale systems integration is necessary, but does not exist currently

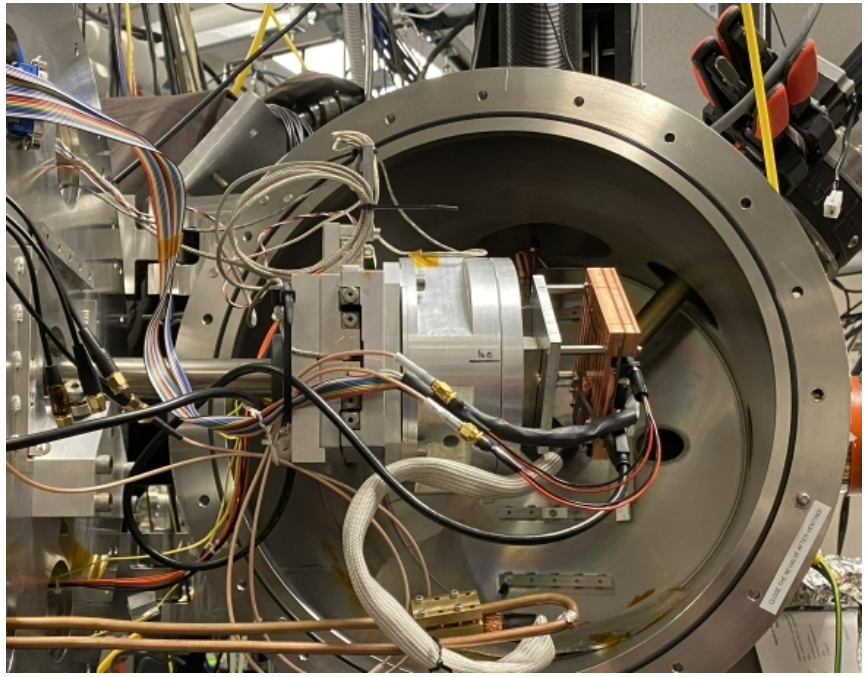
SEE Testing on Processors and Heterogeneous Systems

Few options:

- Fault injection
- Radiation testing (run benchmark, compare to golden), current standard
- Fault emulation/simulation

But there are drawbacks to each technique:

- Fault injection
  - Limited to accessible registers/architecture exposed over the debug port
- Radiation testing
  - Limited insight
  - Developing understanding can be complex and time consuming
- Fault emulation/simulation
  - Abstracted architectural models



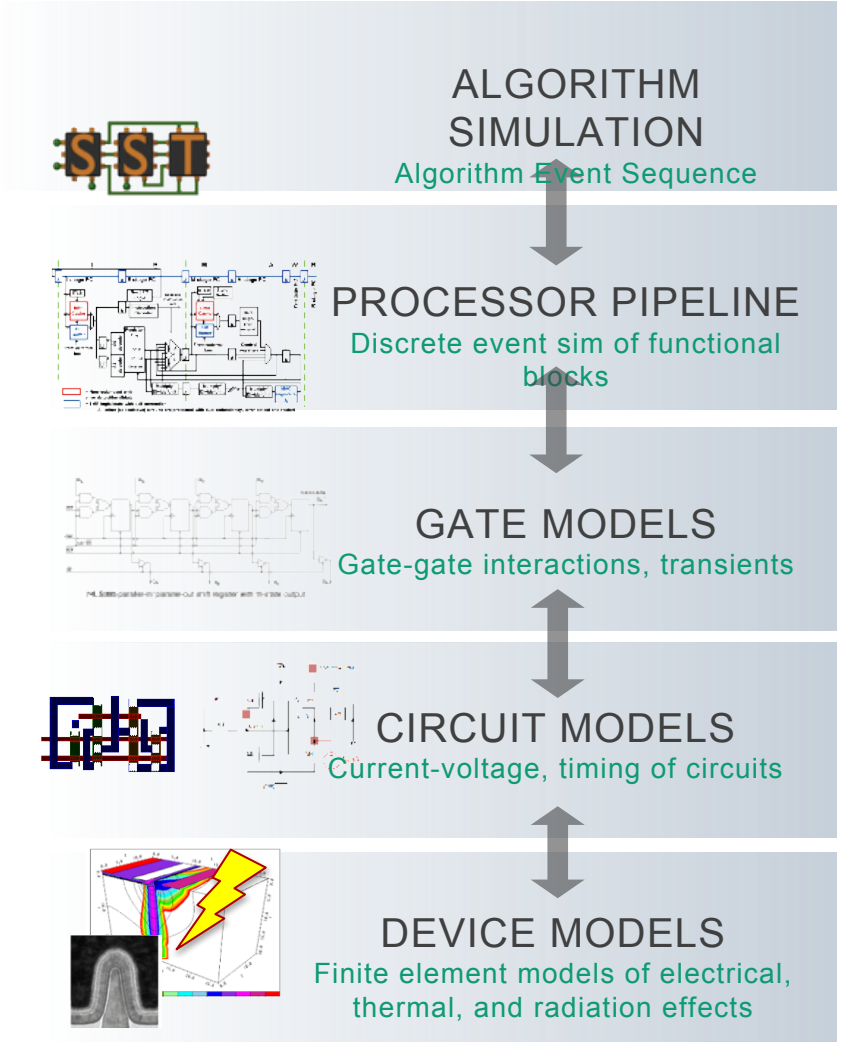
Rad testing of soft-core processor

Proposed Solution

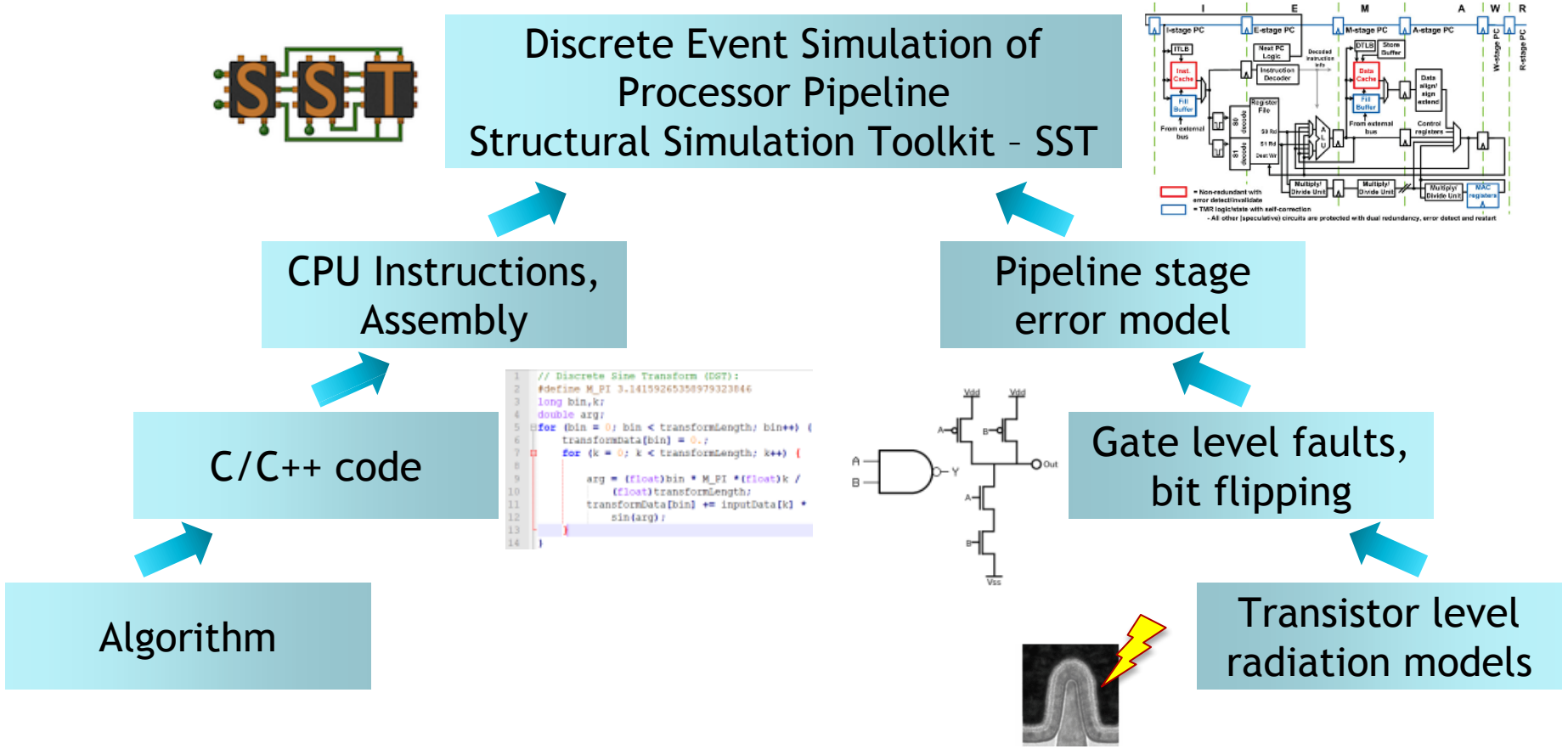
Leverage simulator models of components/modules for reliability testing
- Match registers/logic in hardware to architectural models
- Use new tools to convert synthesized netlists into C simulator code (future work)

Fault injection rates determined by:
- Target technology (e.g. 14nm Fin-Fet)
- Voltage and circuit timing
- Logic masking (from the gates)

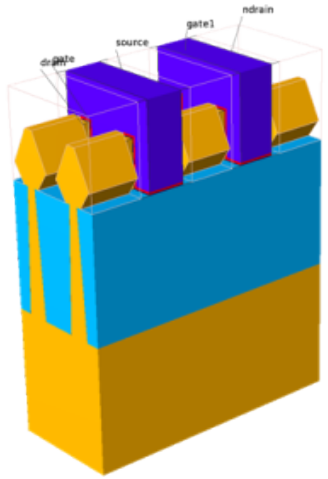
Improvement over previous work
- SEU/SET cross-sections determined from physic models
- Full pipeline model of the processor
- Fault tracking built into register data structures (insight into failure behavior)



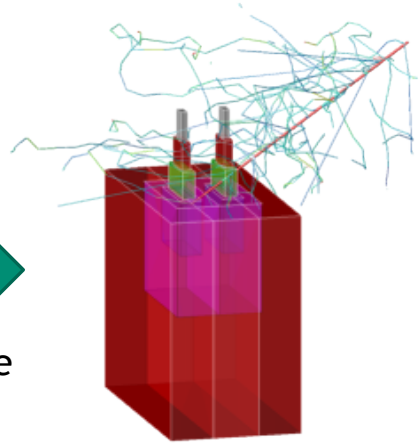
Fault Injection Algorithm and Technology Investigations



Single Event Modeling Framework: TCAD to Circuit

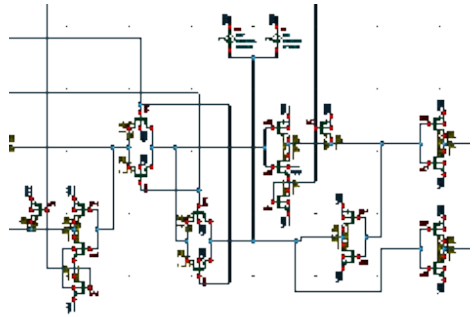


Nested sensitive volume sizes and efficiencies



Collected charge

Critical charge



3D TCAD Simulation

- SEE implemented as a track of charge
- Drift and diffusion charge transport -> determine sensitive volume sizes, locations, and efficiencies

Monte-Carlo Radiative Energy Deposition (MRED) Simulation

- SEE implemented as radiation particles
- Accurately simulates radiative energy deposition in collection volumes derived from TCAD

Circuit Simulation

- SEE implemented as current sources
- Simulates circuit effect based on charge collection determined in MRED custom scripts

Goal is to reduce design cycles and avoid potential layout issues

Case Study: 14 nm DFF MRED Study

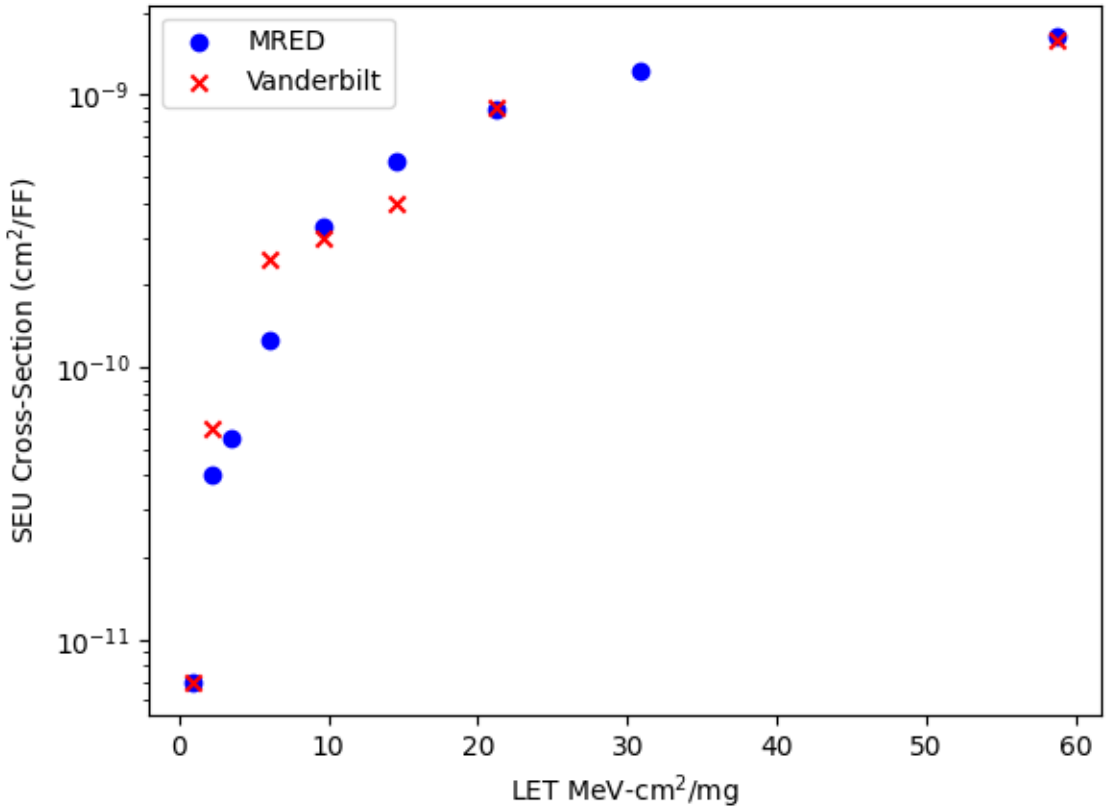
Compute SEU cross-section of 14nm D Flip-Flop vs LET and compare against experimentally measured cross-section (from Vanderbilt University)

- Sample environment is Adams 90% worst-case in an isotropic space environment

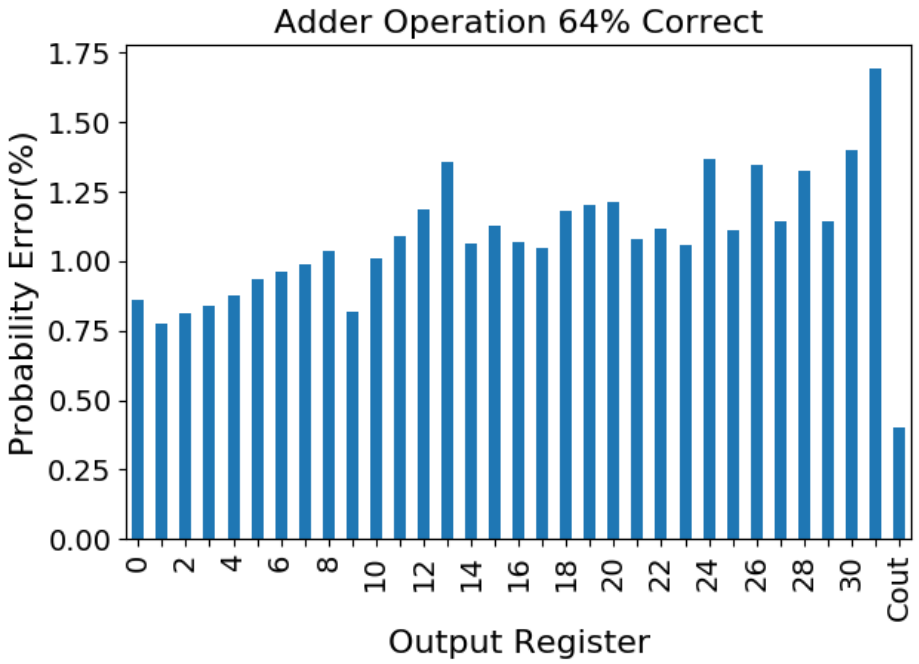
TCAD simulation of a single fin NFET and single fin PFET connected to an inverter

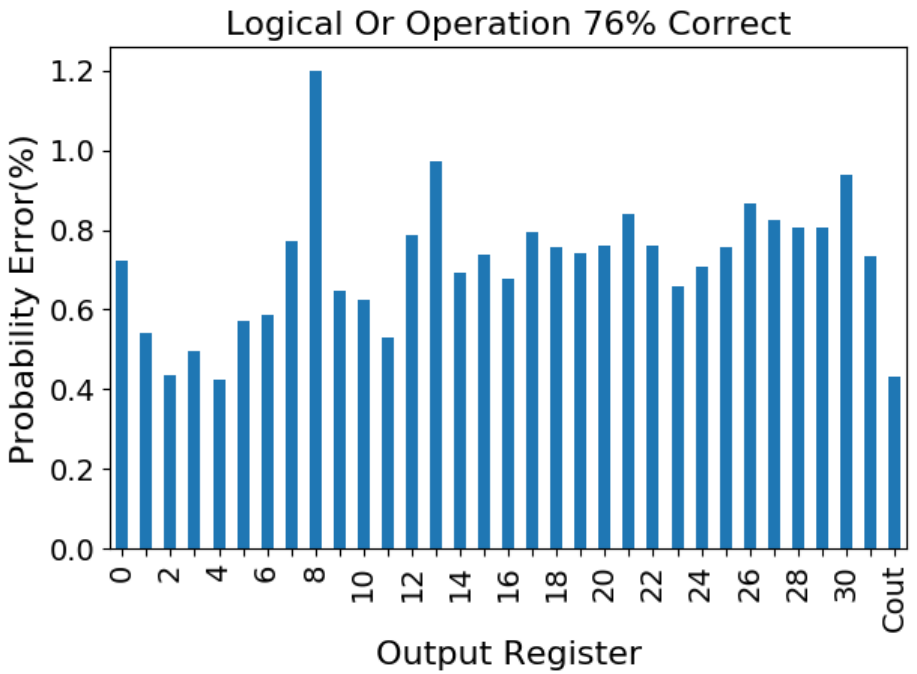
Simulated monoenergetic particles at normal incidence matching test conditions

- Results plotted in figure against experimentally measured data



Logic Masking





Create model of SEU probabilities on the output register of each pipeline stage (but effective cross-section is dominated by the registers themselves and not the combinational logic)

Logical masking can quash some SETs before they are latched as faults

Figures considers logic masking on a 32-bit RISC ALU

- Assume SET on each gate has the same probability
  - Add (left) and or (right) shown

Microprocessor & Fault Simulation

We employ a high performance computing simulator called SST (http://sst-simulator.org/)

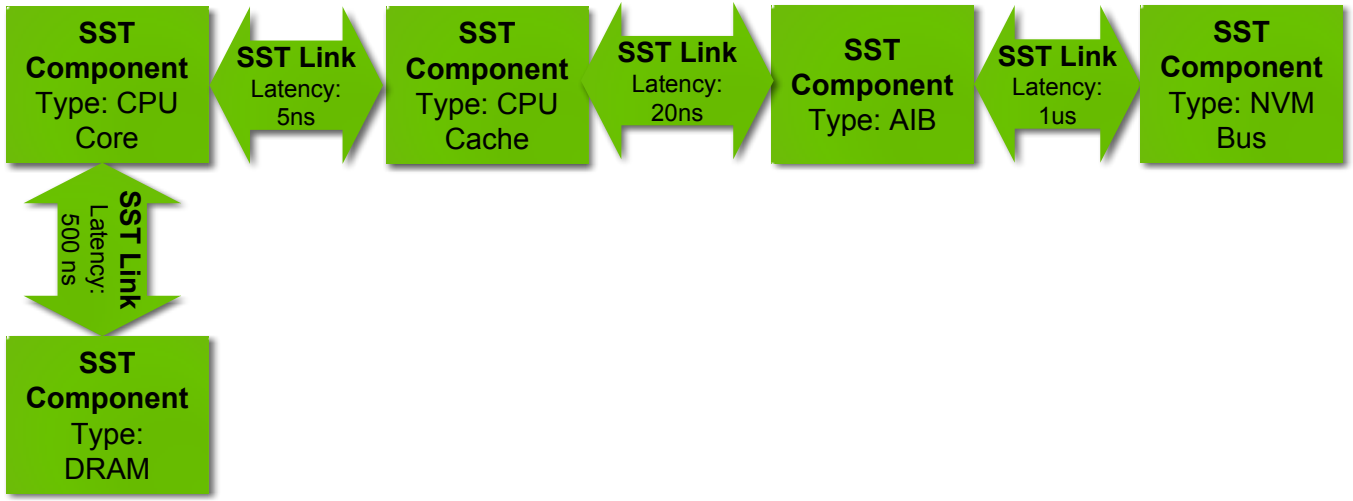
- Uses “components” connected by links

Modified to allow fault injection & tracking

- Track current (maybe faulty) data
- Track “correct” data

Fault can be tracked for impact

- Was it quashed (and how)?
- How far does it spread?
- 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

Case Study : HERMES Processor

Radiation hardened by design

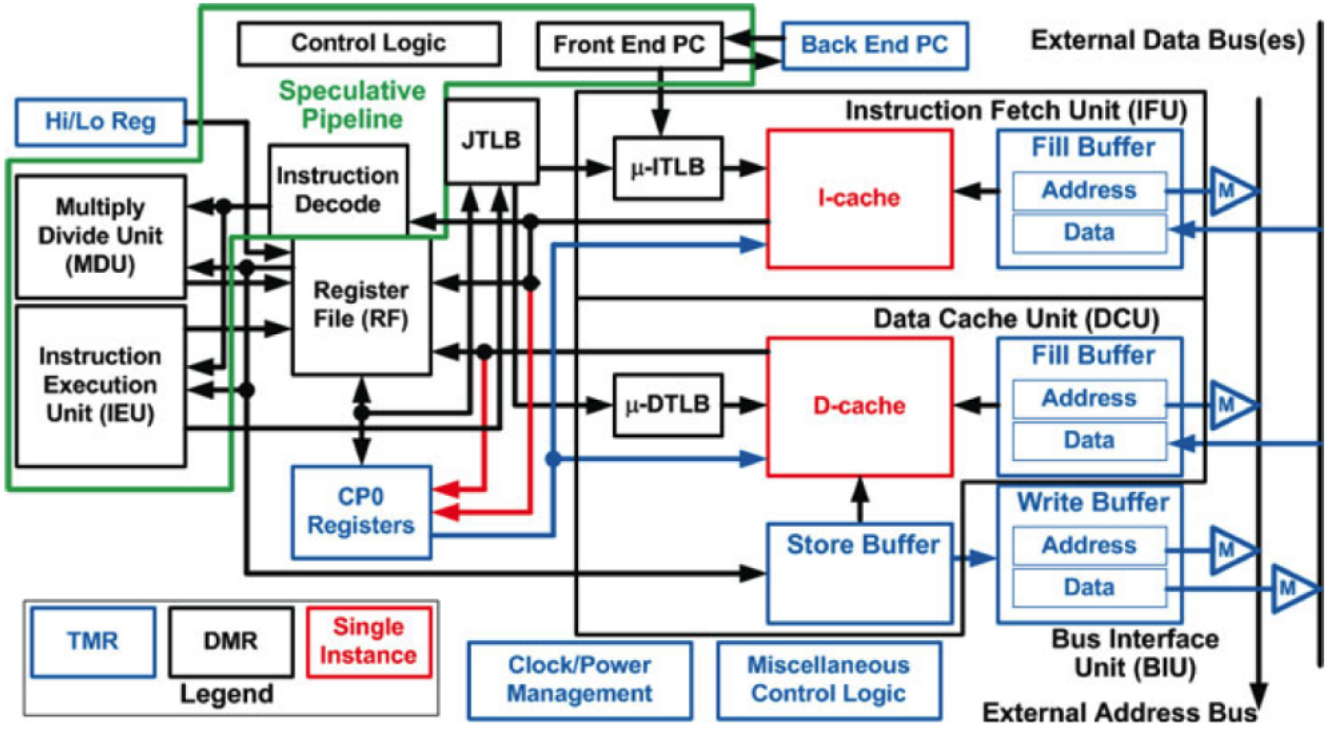
Faster and lower energy then triple redundancy, reports errors to algorithm

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)



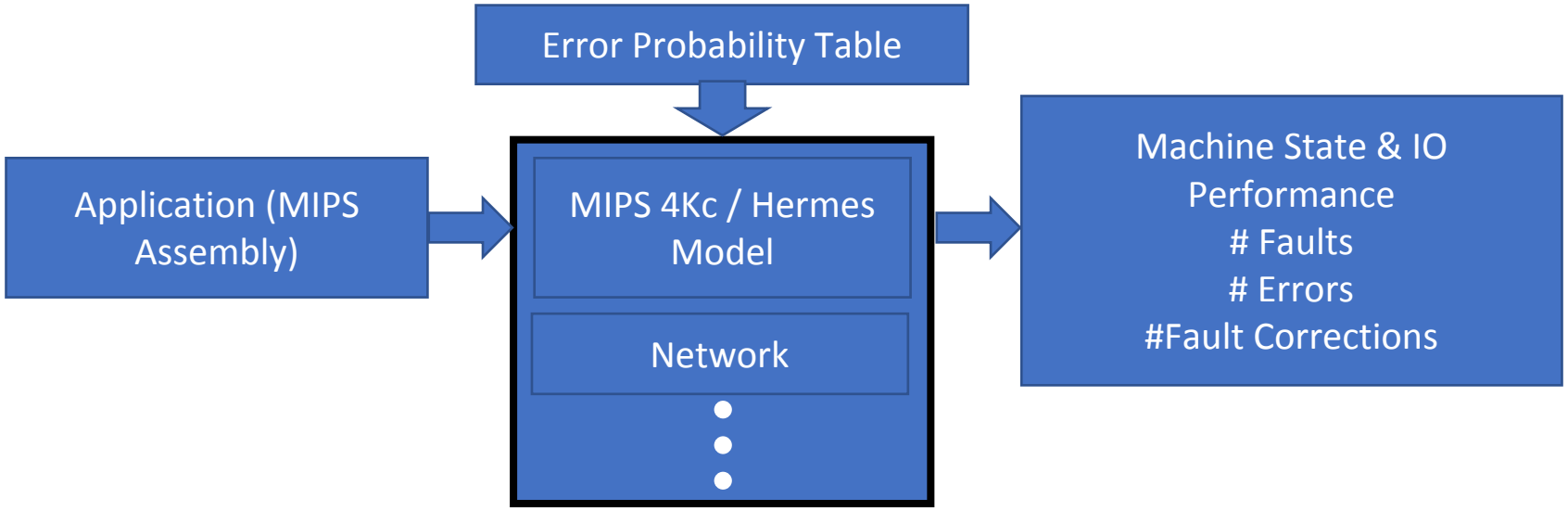
Arizona State University

Fault Injection Capabilities

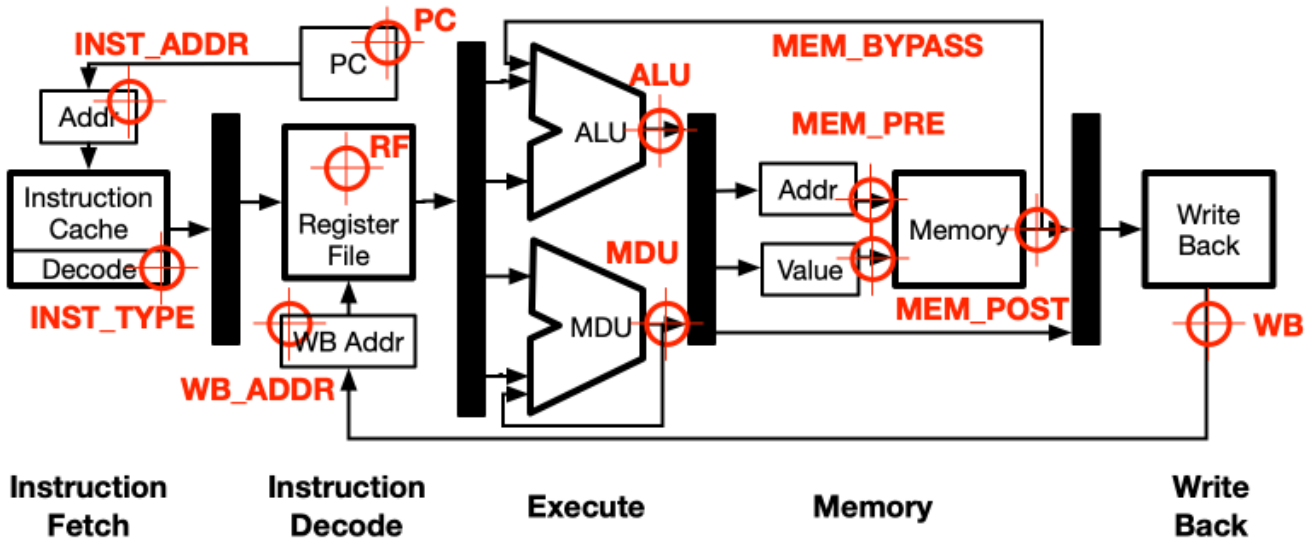
Can inject faults at the beginning of every clock cycle
- Randomly for form a precomputed table (for repeatability)
  - Precomputed table is generated randomly based on the environment

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



Register Modeling



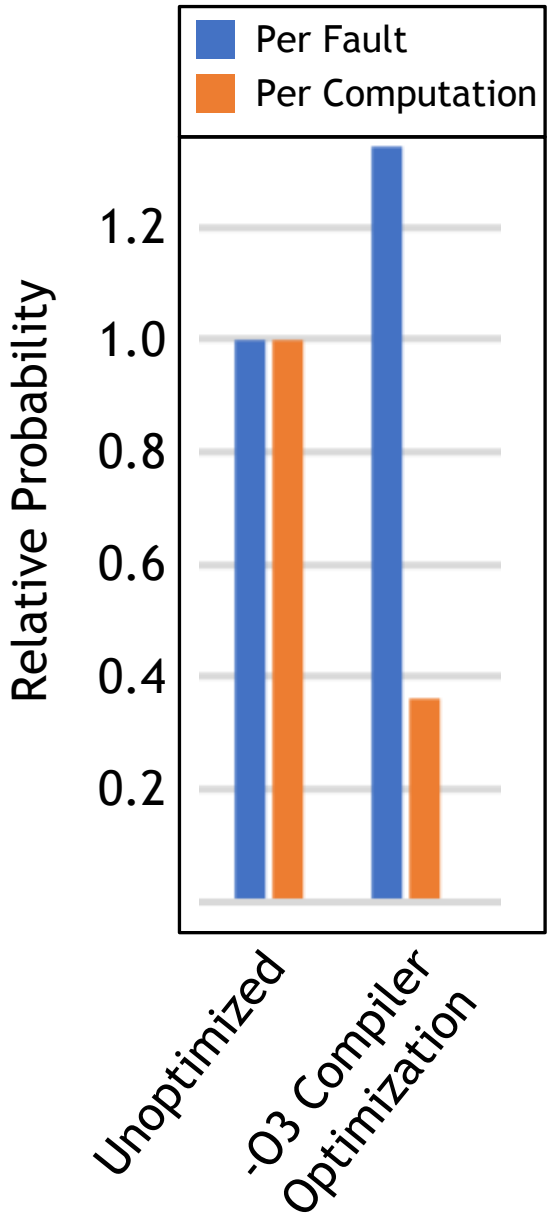
Location	Bit	Location	Bit
ALU	64	CONTROL	52
INST_ADDR DATA	40	INST_ADDR CTRL	16
INST_TYPE DATA	32	INST_TYPE CTRL	5
MDU DATA	233	MDU CTRL	122
MEM_POST DATA	160	MEM_POST CTRL	8
RF DATA	992	RF CTRL	5
MEM_PRE_ADDR DATA	203	MEM_PRE_ADDR CTRL	48
MEM_PRE_DATA DATA	256	MEM_PRE_DATA CTRL	8
MEM_PRE_ADDR BYTE	16	MEM_PRE_DATA BYTE	23
MEM_BP_VAL DATA	32	WB	32
PC	64		

Software Benchmarks

Matrix Multiply (12x12 w/ 32-bit unsigned integers)
- Uses triple-nested loop

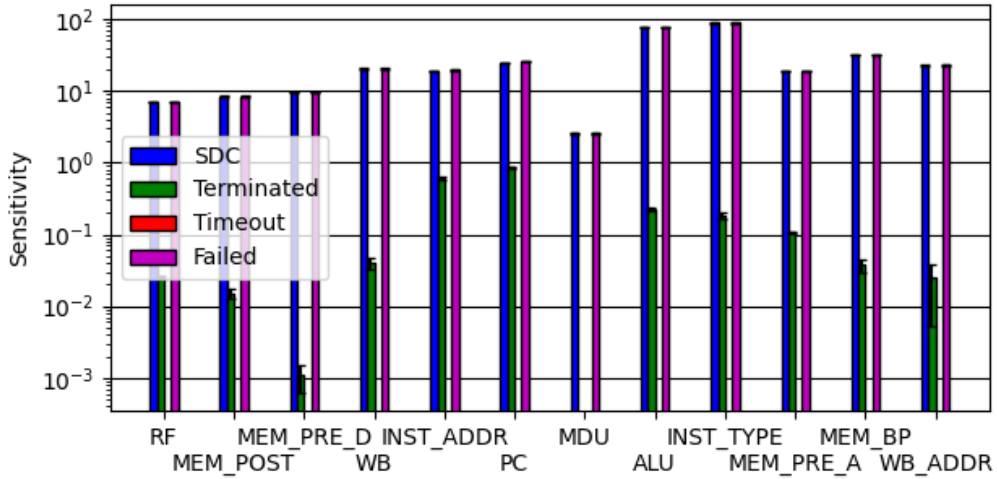
Variations of MM used
- Compiler optimizations (O3)
- Software redundancy (DMR, TMR)

Initial findings demonstrate expected results
- Optimizations make each instruction more vulnerable
- But optimizations make program less vulnerable (fewer instructions/faster execution)

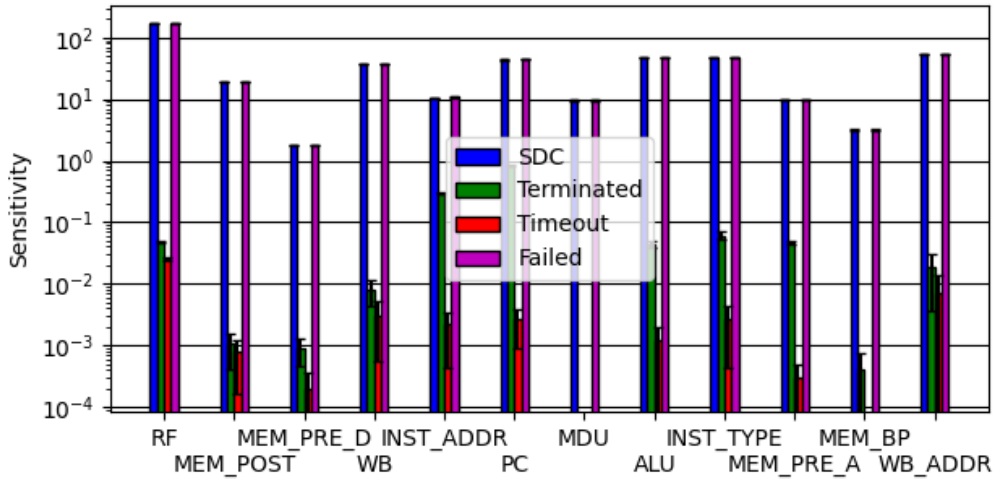


Fault Injection Results

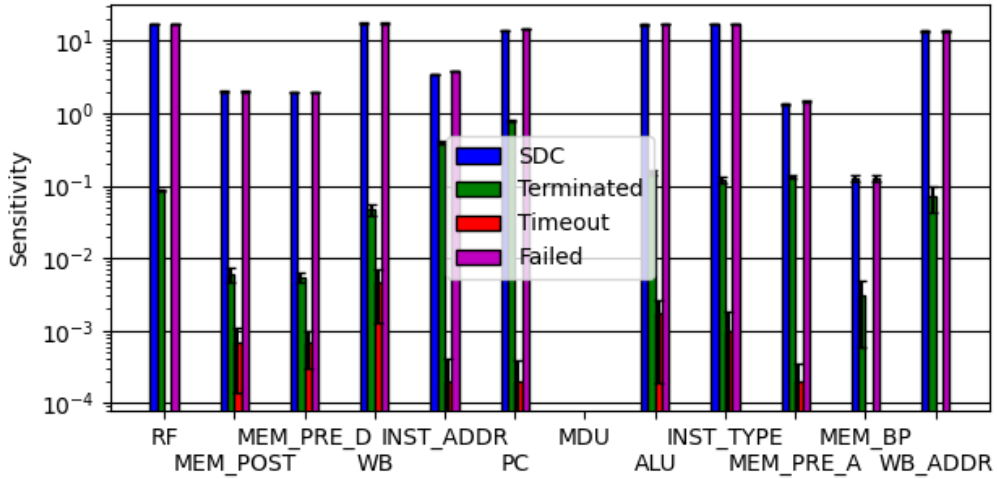
Matrix Multiply



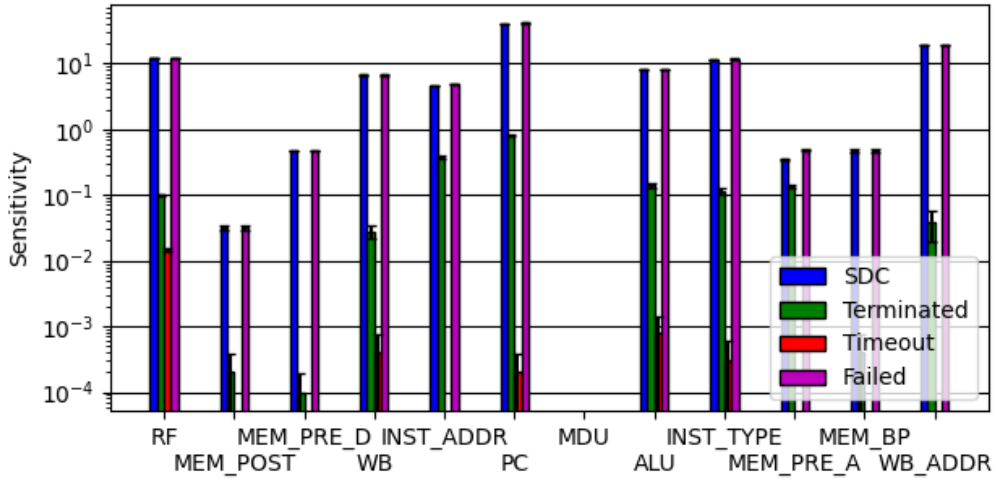
Matrix Multiply O3



DMR O3

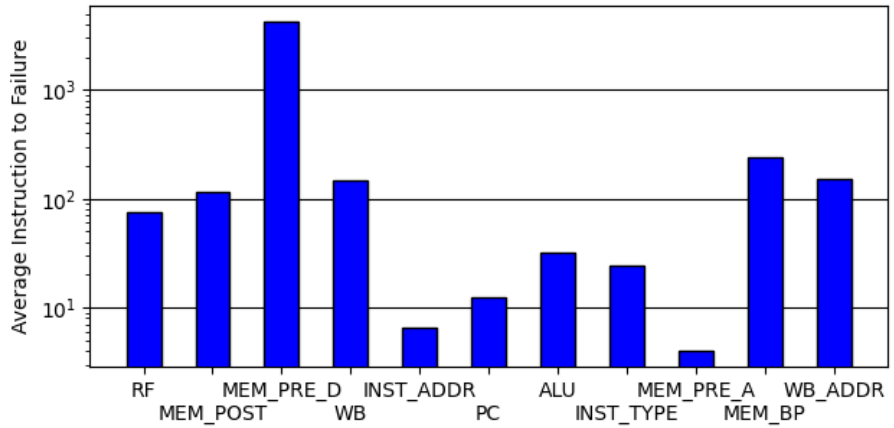


TMR O3

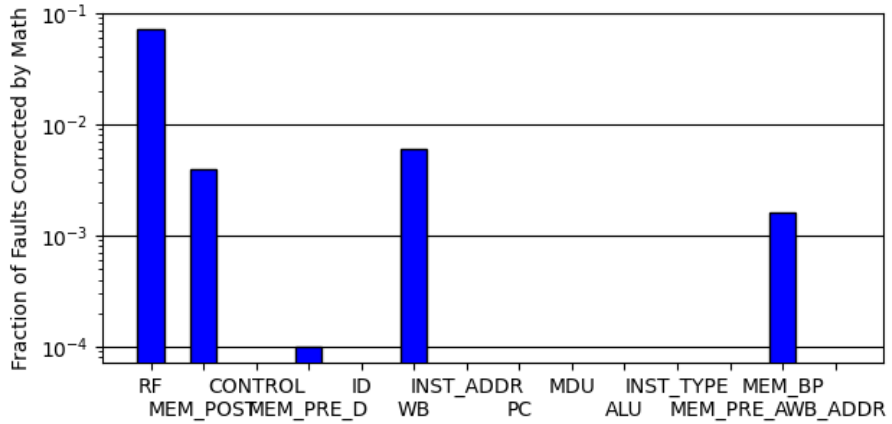


Other Fault Injection Statistics

Avg. Inst. To Failure



Math Corrections



Average Instruction to Failure

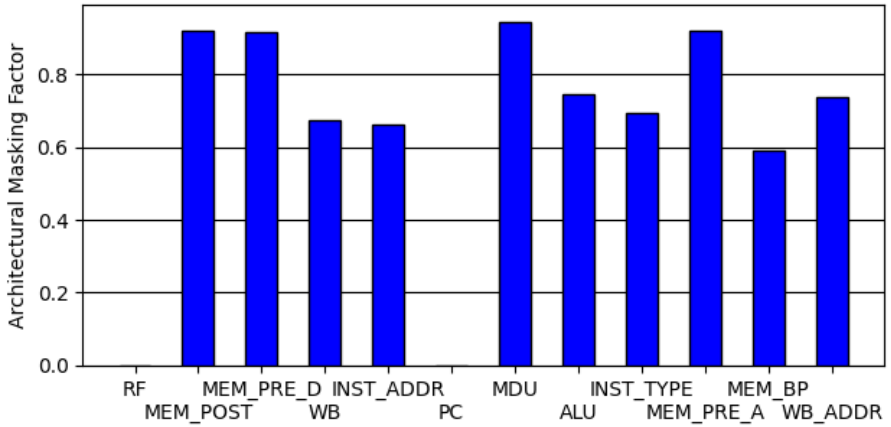
- How many instruction executions until a failure was observed after a fault.
- Mem pre data took longest to fail
- Mem pre address failed quickly after injection

Math Corrections

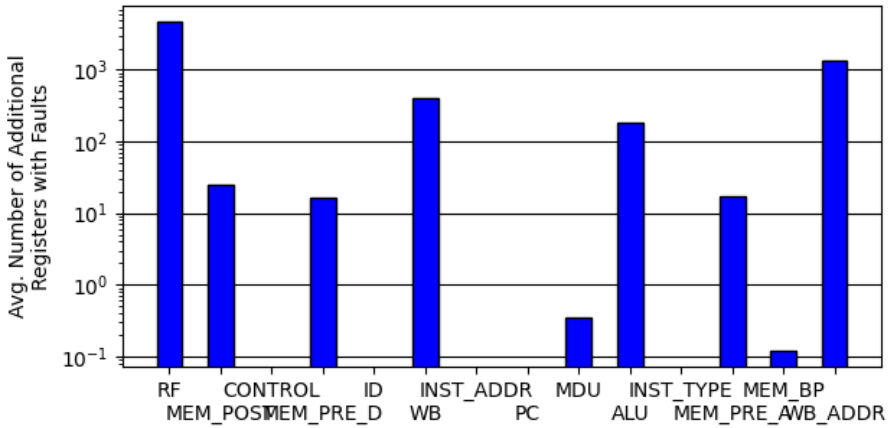
- Fraction of faults corrected by math operations
- RF faults most likely to get corrected during math operations executed during normal program execution

Other Fault Injection Statistics

Faults into unused Components



Fault Spreading



Faults into unused components

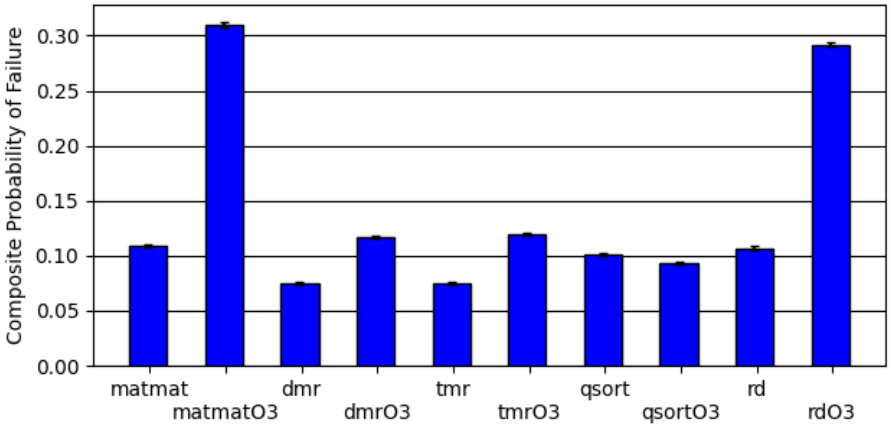
- Faults into registers that were not in use during that clock cycle
- Faults had no impact on program execution

Fault Spreading

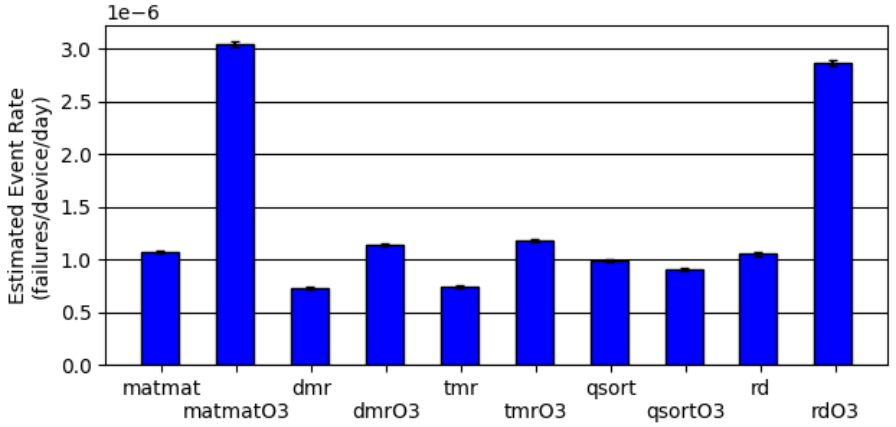
- Number of registers corrupted at end of program execution (either successfully or not)
- RF faults spread to many other registers
- Some faults do not spread (likely to immediate program termination)

Estimated Event Rates

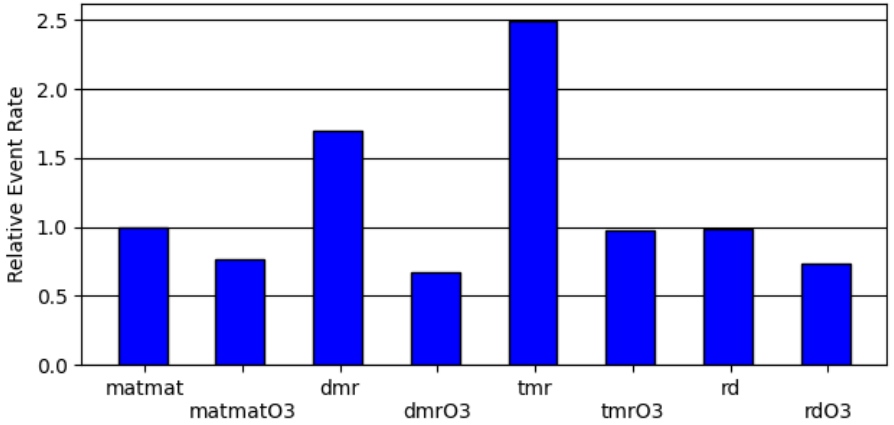
Probability of Failure (given SEU)



Event Failure Rate



Relative Event Failure Rate



Composite probability of failure can be obtained from:

- Each registers probability of failure
- Number of bits per register

Probability of failure can be used to calculate event rate

- Using TCAD/MRED calculations
- Assuming the algorithm runs continuously

Relative failure rates can be calculated using each versions runtime (in clock cycles)

- DMRO3 performs best (reduces clock cycles while adding

Conclusion

There is a need for insight into how processors fail in harsh environments
- Minimal insight into architectural state with just JTAG/Debug port access

Our model lets us see architectural state and provides more insight into why some algorithms perform better than others
- DMR O3 had best relative rate due to decreasing RF errors (DMR) and reducing clock cycles (O3)

Device level physics data enables us to estimate more realistic cross-sections for the processor and each algorithm

Future Work

Create simulation model directly from synthesized netlist
- Simulation models already exist for post-synthesis debug – extend to allow fault tracking capabilities
- Tradeoff in hardware fidelity and simulation performance/speed

Perform radiation test on HERMES processor and compare results
- 14nm version of HERMES built
- HERMES has built in commands to allow insight into the internal state of the processor