Fast and Accurate Vulnerability Analysis of a RISC-V Processor

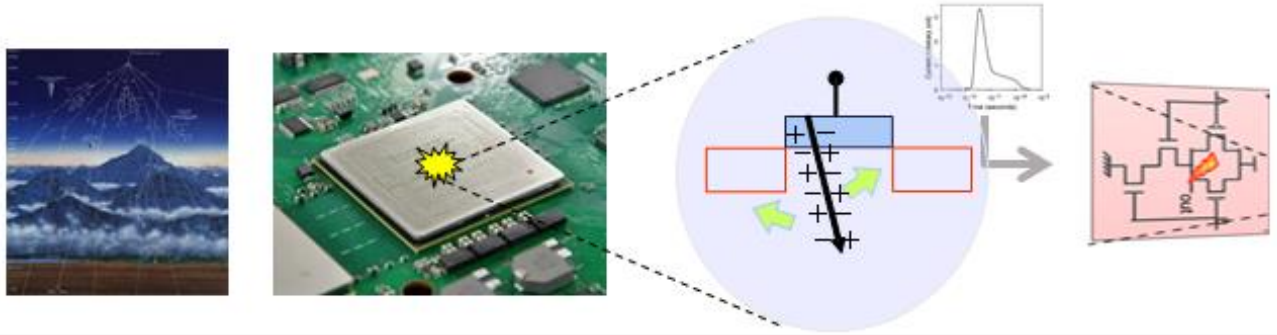
Joseph Paturel, Simon Rokicki, Olivier Sentieys

Univ. Rennes, Inria, IRISA

Why care about Fault Tolerance?

- Modern technologies
  - Lower node capacitances
  - Denser layouts
  - Increased frequencies
- Energy efficiency
  - Lower supply and threshold voltages

High SET sensitivity



Vulnerability Analysis

- Fault injection, simulation or emulation most often:
  - Only injects single-bit faults
  - Does not model the microarchitecture
  - Ignores combinational logic
- Memory/register fault injection is not enough
  - Need to model microarchitecture
  - Need to consider combinational logic [1]
- New technologies exhibit multi-bit error behaviors
  - Need to model MBUs as well as SEUs

[1] N. N. Mahatme *et al*, «Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process», *IEEE Trans. On Nuclear Science*, Dec. 2011

Contributions

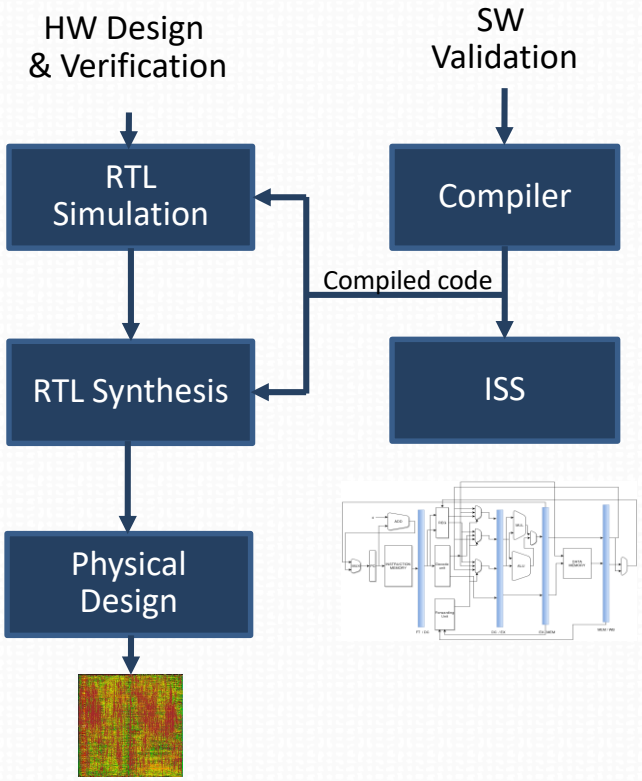
- MBUs are present and are here to stay
- Fault injection methodology and flow (Part II)
  - From gate to microarchitecure
  - MBU-aware
  - Fast and accurate
- Use case: Comet RISC-V processor core (Part I)

Part I: Comet

a HLS designed RISC-V Core

What You Simulate is What You Synthesize

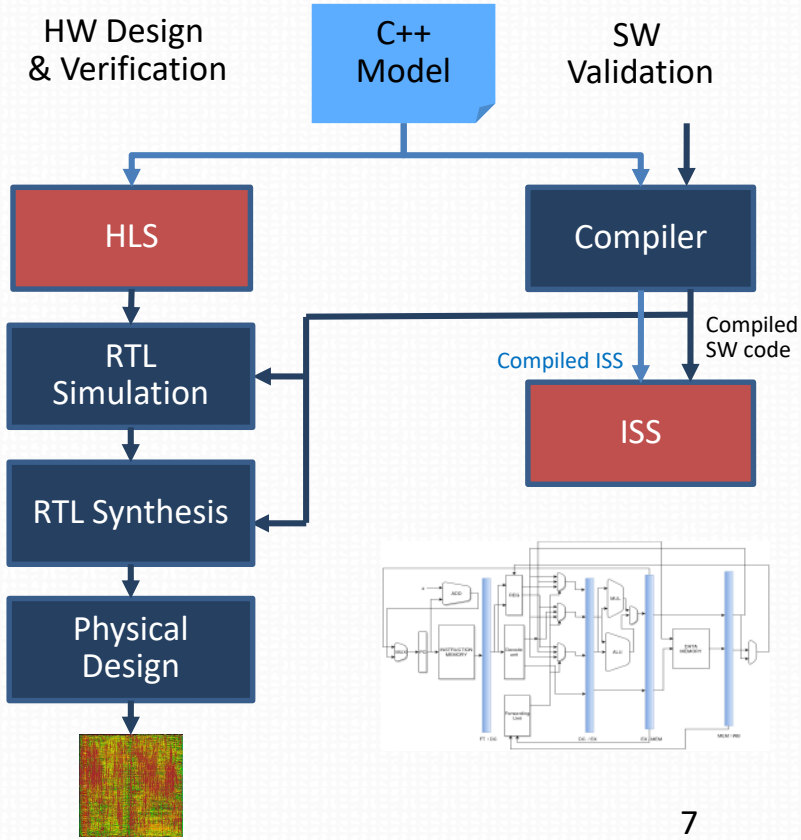
- Traditional Processor Design Flow
  - Maintain two coherent models:
    - RTL and simulation (ISS) models



What You Synthesize is What You Simulate

- Traditional Processor Design Flow
  - Maintain two coherent models:
    - RTL and simulation (ISS) models

- Proposed Flow
  - Design the processor as well as its software validation flow from a single high-level model



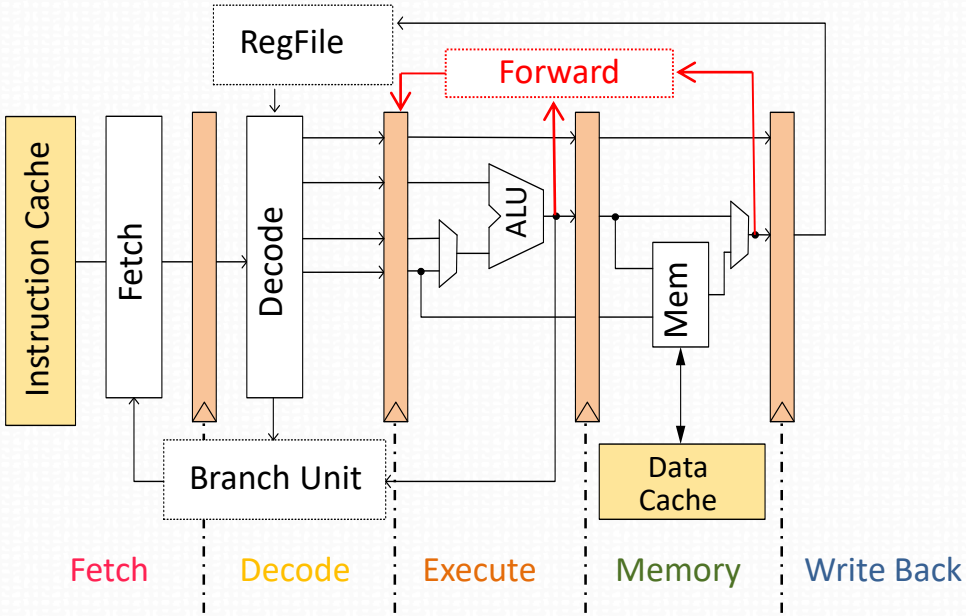
Explicitly Pipelined Simulator (1/2)

- Comet core
  - 32-bit RISC-V instruction set RV32IM
  - In-order 5-stage pipeline micro-architecture
- Pipelined stages are explicit
- Main loop is pipelined (II=1)

- Explicit stall mechanism
- Explicit forwarding

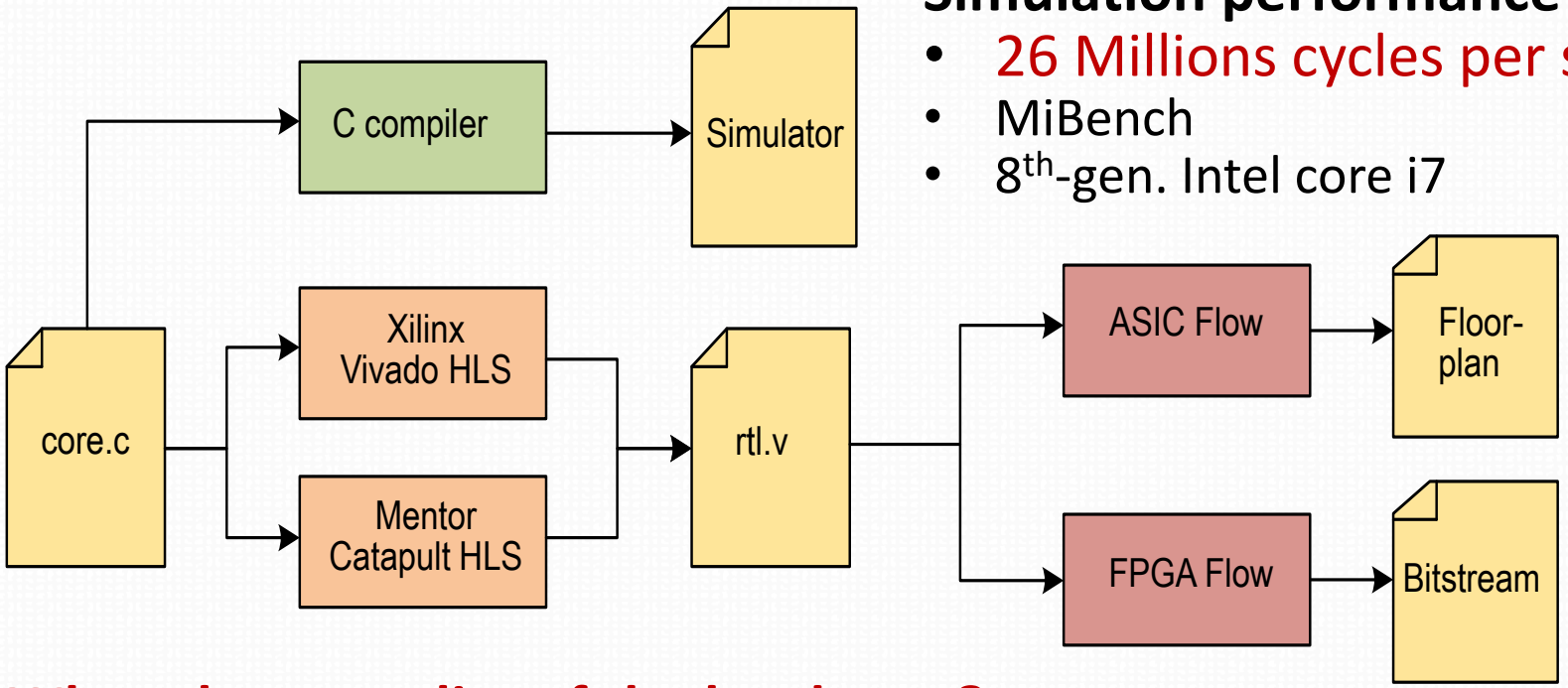
```
struct FtoDC ftodc;
struct DCtoEx dctoex;
struct ExtoMem extomem;
struct MemtoWB memtowb;
while true do
    ftodc_temp = fetch();
    dctoex_temp = decode(ftodc);
    extomem_temp = execute(dctoex);
    memtowb_temp = memory(extomem);
    writeback(memtowb);
    bool forward = forwardLogic();
    bool stall = stallLogic();
    if !stall then
        ftodc = ftodc_temp;
        dctoex = dctoex_temp;
        extomem = extomem_temp;
        memtowb = memtowb_temp;
    end
    if forward then
        dctoex.value1 = extomem.result;
    end
end
```

Explicitly Pipelined Simulator (2/2)




```
struct FtoDC ftodc;
struct DCtoEx dctoex;
struct ExtoMem extomem;
struct MemtoWB memtowb;
while true do
    ftodc_temp = fetch();
    dctoex_temp = decode(ftodc);
    extomem_temp = execute(dctoex);
    memtowb_temp = memory(extomem);
    writeback(memtowb);
    bool forward = forwardLogic();
    bool stall = stallLogic();
    if !stall then
        ftodc = ftodc_temp;
        dctoex = dctoex_temp;
        extomem = extomem_temp;
        memtowb = memtowb_temp;
    end
    if forward then
        dctoex.value1 = extomem.result;
    end
end
```

Design and Validation Flow



Simulation performance

- 26 Millions cycles per sec.
- MiBench
- 8th-gen. Intel core i7

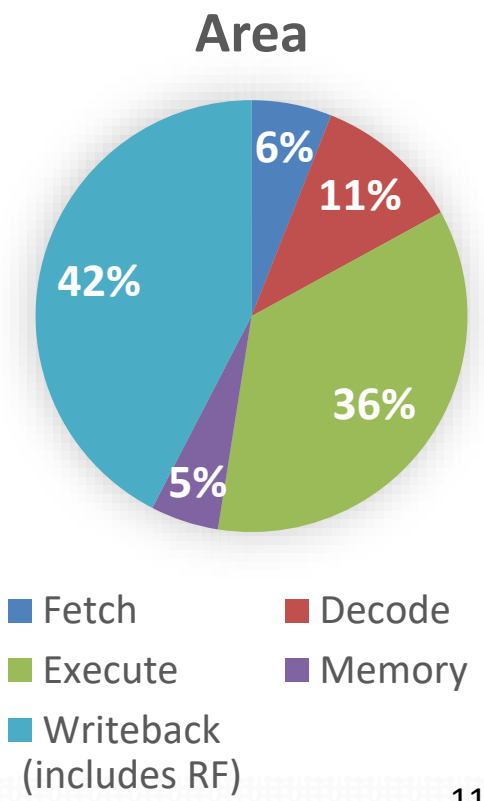
What about quality of the hardware?

Synthesis Results

- Target technology is STMicro 28nm FDSOI
- All cores are configured for rv32i

AREA AND FREQUENCY RESULTS FOR DIFFERENT RISC-V CORES.

Core	Language	Frequency Target (MHz)	Area (μm^2)
Comet [1]	C++	700	8 476
PicoRV32 [3]	Verilog		7 830
Rocket [4]	Chisel		11 764



Advantages and Limitations

Advantages

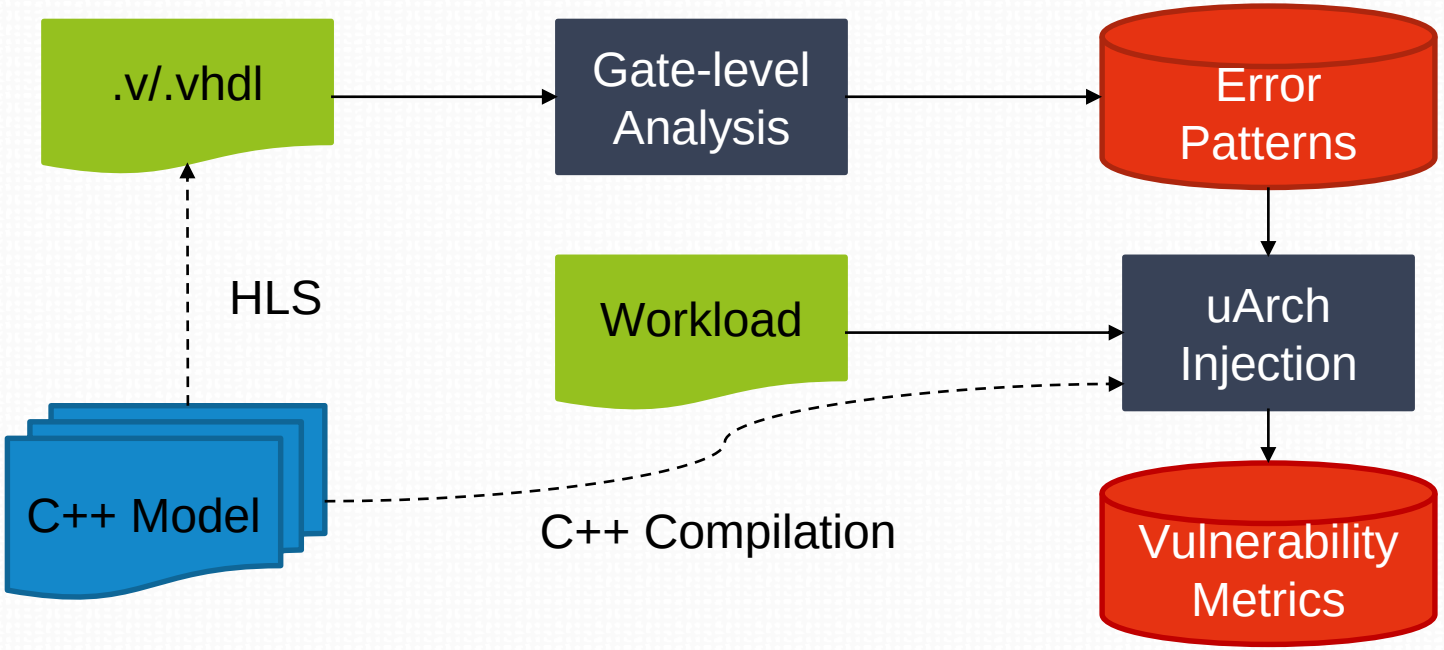
- Improves readability, productivity, maintainability, and flexibility of the design
- Fast simulation (~20.10^6 cycles/s)
- Object-Oriented processor model can be easily modified, expanded and verified

Limitations

- Pipeline stages and some features (e.g. multi-cycle operators) have to be explicit
- HLS tools may have trouble synthesizing large multi-core systems…

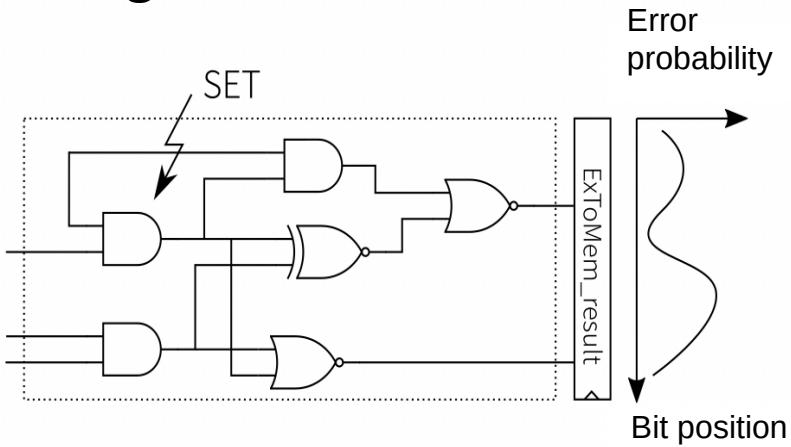
Part II: Vulnerability Analysis Flow

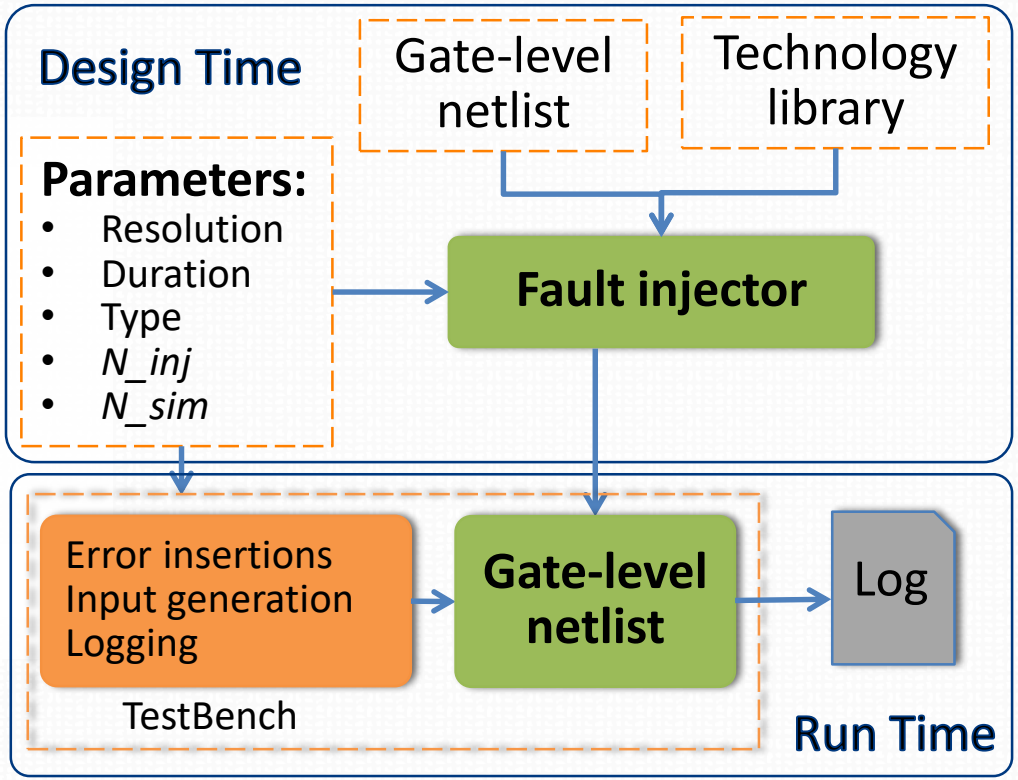
Proposed Approach to Vulnerability Analysis



1/ Gate-level Analysis

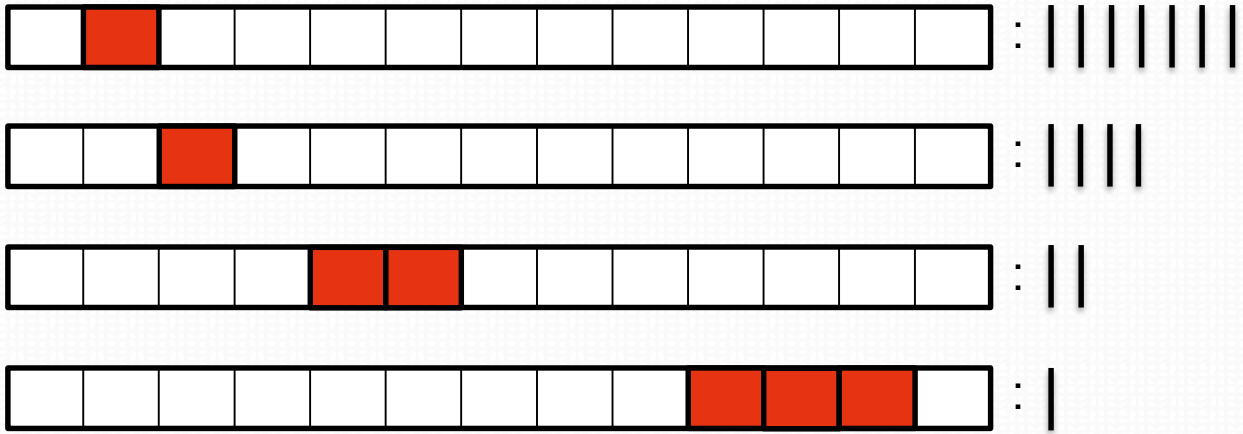
- Inject SETs in the gate-level netlist





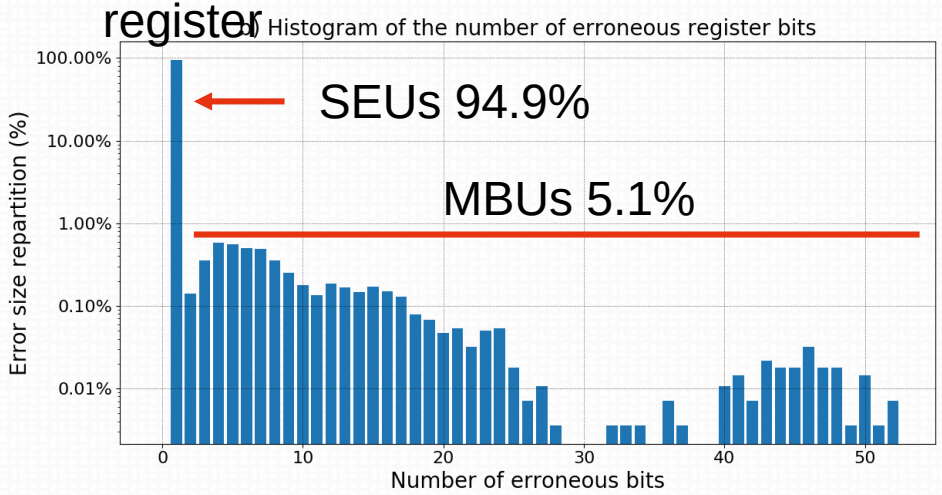
1/ Gate-level Analysis

- Logging patterns and error probability (SEUs + MBUs)



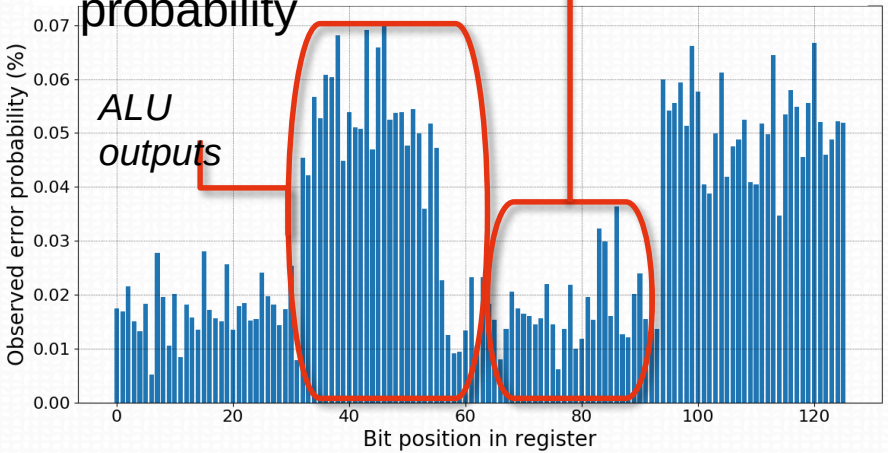
Results: Comet Execution Stage

Number of erroneous bits in output register



Dest. Register, Opcode Forwarding, etc.

Output register per bit error probability



1 Million injections

Influence of SET Width and Frequency on MBUs

- Fixed width (400ps)

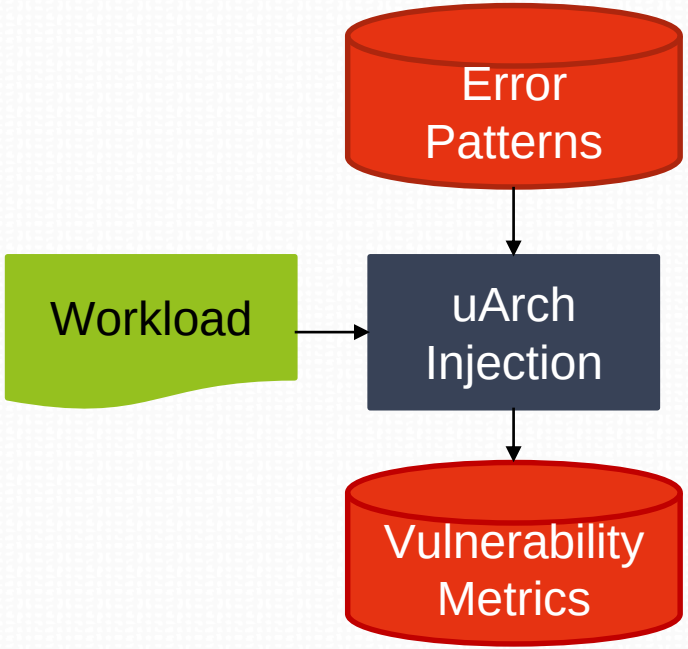
Freq.	200 MHz	300 MHz	400 MHz	500 MHz	600 MHz
SEU	9,308 93%	15,592 96.3%	23,613 94.1%	26,489 94.9%	30,919 95.5%
MBU	699 7%	599 3.7%	1,473 5.9%	1429 5.1%	1,447 4.5%

- Fixed frequency (500MHz)

SET	100 ps	200 ps	400 ps	500 ps
SEU	5,144 97.6%	10,529 95.3%	26,489 94.9%	33,449 95.9%
MBU	127 2.4%	755 4.7%	1429 5.1%	1,432 4.1%

2/ Microarchitectural-Level Fault Injection

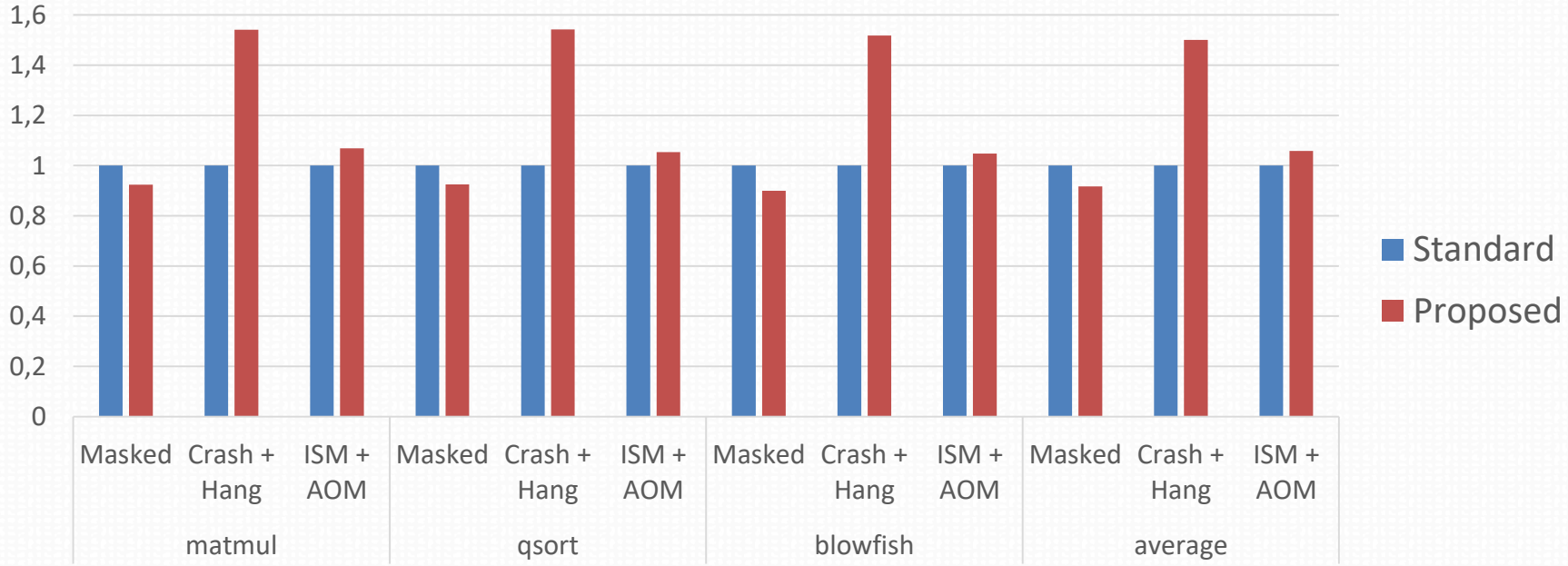
- Augmented simulator allows for injection of gate-level fault patterns
- Injection is guided by the area of the different pipeline stages
- Fault classes considered:
  - Crashes and Hangs
  - ISM, AOM, ISM & AOM



ISM: Internal State Mismatch
AOM: Application Output Mismatch

Comet Vulnerability Analysis Results

- Error class proportions
- Standard vs. proposed approach



Conclusion on Vulnerability Analysis

- MBUs are present and are here to stay
- MBUs significantly impact AVF
  - more than 50% critical errors (crashes & hangs)
- Fault injection methodology and flow
  - From gate to microarchitecure
  - Conscious of MBU patterns and error probability
  - Fast and accurate

Conclusion & Roadmap on Comet

- Efficient processor core design (HW µarch + SW simulator) from a single C++ code
- Current projects
  - Dynamic Binary Translation, Non-Volatile Processor, Fault-Tolerant Multicore, etc.
- Perspectives
  - Automatic source-to-source transformations for HLS
    - From ISS-like specification to HLS-optimized C code
  - Support for floating point extension
  - RTOS Support (process, interrupt controller, peripherals)
  - Multi-core system with cache coherency (Q4 2019)
  - Many-core system with NOC (2020)

Questions

Thank you for your attention!

https://gitlab.inria.fr/srokicki/Comet