embeddedworld2017
Exhibition&Conference
... it´s a smarter world

MULTICORE DESIGN SIMPLIFIED
imperas

Fast Fault Injection to Evaluate Multicore Systems Soft Error Reliability

Felipe Rosa[1], Luciano Ost[2], Ricardo Reis[1], Simon Davidmann[3], Larry Lapides[3]

[1]UFGRS – Instituto de Informatica – PGMicro/PPGC
[2]University of Leicester, Dept. of Engineering
[3]Imperas Software Ltd.

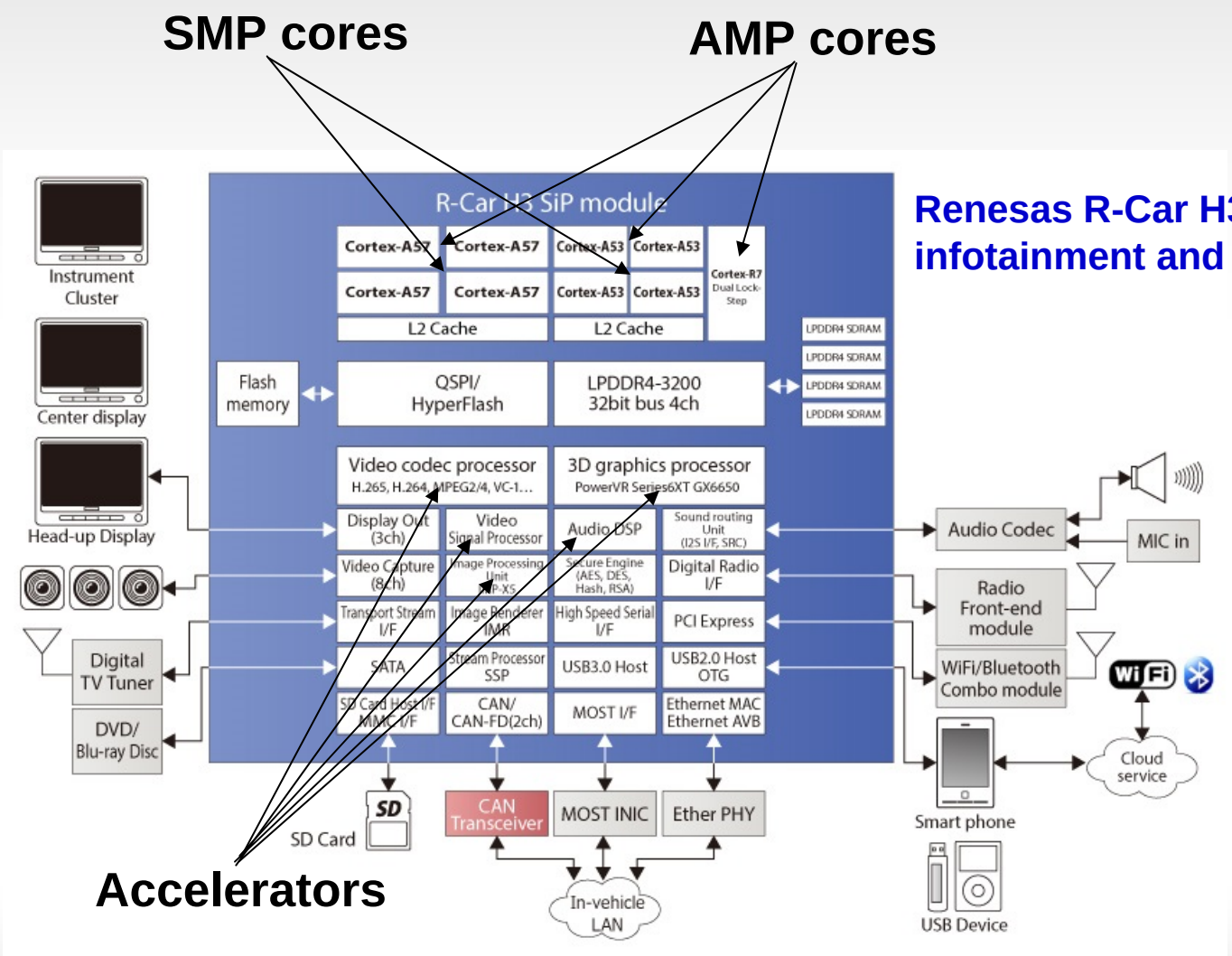
Agenda

MULTICORE DESIGN SIMPLIFIED
imperas

- Why is fault injection testing needed?
- Current testing methodology
- Simulation alternatives
- Instruction accurate vs cycle accurate simulation experiment
- Results
  - Simulation performance
  - Fault injection results
- Conclusions

Modern SoCs Have Many Concurrent Processing Elements

MULTICORE DESIGN SIMPLIFIED
imperas



Renesas R-Car H3: automotive infotainment and ADAS applications

Current SoCs and Systems are Susceptible to Soft Errors

MULTICORE DESIGN SIMPLIFIED
imperas

- Soft errors are typically caused by radiation
- Also called Single Event Effects (SEEs) or Single Event Upsets (SEUs)
- Can get as many as 280 soft errors per day for spacecraft; 1 per day for ground systems
- Systems are becoming more susceptible because of increases in the overall complexity of the devices and software
  - Multiple cores
  - Aggressive clock rates
  - Multiple voltage domains
- Risk is human life, or high financial cost to company: risks are huge

Fault Tolerance Testing is Needed

MULTICORE DESIGN SIMPLIFIED
imperas

- Compliance with industry safety standards requires products to demonstrate reliability and tolerance to fault injections
  - For example: Automotive ISO26262 requires this
- Companies also want to optimize architectures to achieve fault tolerance
- Fault injection testing currently performed using hardware and low-level RTL simulation
- Can software simulation be used to supplement and accelerate fault injection testing?

Current Techniques for General Embedded System Testing

MULTICORE DESIGN SIMPLIFIED
imperas

- Hardware based testing
  - Actual production hardware
  - Development boards or other hardware prototypes
  - Hardware emulators
- Cycle accurate simulation
- Instruction accurate simulation

- Hardware based testing is the norm
- Cycle accurate simulation is too slow
- Instruction accurate simulation has advantages of controllability, observability, determinism, ease of automation
  - Now starting to move into mainstream as a complementary tool to hardware based testing

Hardware-Based Software Testing

MULTICORE DESIGN SIMPLIFIED
imperas

- Has timing/cycle accuracy
- JTAG-based debug, trace
- Traditional development board / emulation based testing
  - Limited physical system availability
  - Limited external test access (controllability)
  - Limited internal visibility

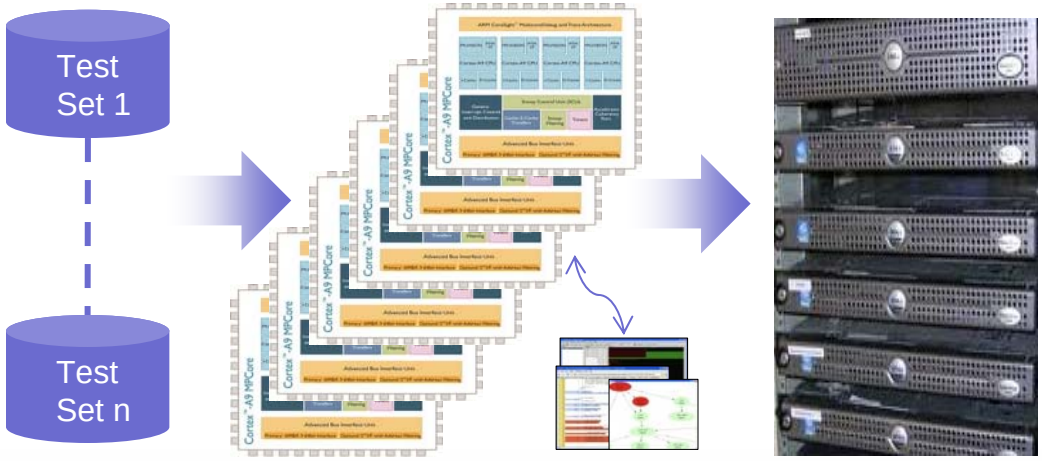
- To get around these limitations, software is modified
  - printf
  - Debug versions of OS kernels
  - Instrumentation for specific analytical tools, e.g. code coverage, profiling

- Modified software may not have the same behavior as clean source code

Advantages of Virtual Platform Based Software Development

MULTICORE DESIGN SIMPLIFIED
imperas

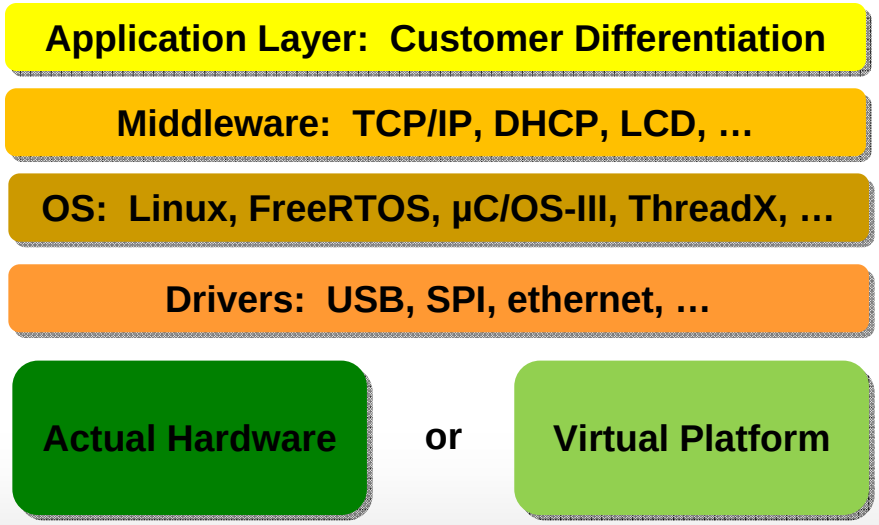
- Earlier system availability
- Full controllability of platform both from external ports and internal nodes
  - Corner cases can be tested
  - Errors can be made to happen
- Full visibility into platform: if an error occurs, it will be observed by the test environment
- Easy to replicate platform and test environment to support automated regression testing on compute farms



Virtual Platforms Complement Hardware-Based Software Development

MULTICORE DESIGN SIMPLIFIED
imperas

- Current methodology employs testing on hardware
  - Proven methodology
  - Has limitations
  - We are at the breaking point
- Virtual platform based methodology delivers controllability, visibility, repeatability, automation



Virtual platforms – software simulation – provide a complementary technology to the current methodology

Current Techniques for Fault Injection Testing

MULTICORE DESIGN SIMPLIFIED
imperas

- Fault injection test methodology is following a similar evolutionary path to general embedded software testing, but maturity of methodology is lagging by a few years

- Hardware based testing is currently the norm, but is expensive, difficult to automate, difficult to get the fault coverage (only “blackbox” testing), difficult to get the required observability and is limited by the architecture of the specific device
- RTL simulation based testing is slow (10^4 times slower than real time) and lacks access to a reasonable range of processor models
- Cycle accurate simulation is slow and expensive, but might provide a more comprehensive environment
- **Instruction accurate simulation is fast, less expensive, provides a good testing environment, but does it provide the required accuracy?**

Experiment Plan

MULTICORE DESIGN SIMPLIFIED
imperas

- Test instruction accurate simulation accuracy by comparing with cycle accurate simulation results

- Use gem5 for cycle accurate simulator
  - Open source, event based simulator
- Two simulation modes:
  - “gem5 Detailed” simulator has no approximations, uses all the details available for cycle accurate simulation
  - “gem5 Atomic” simulator emulates memory and cache behavior

- Use Open Virtual Platforms (OVP) OVPsim for instruction accurate simulator
  - Typical performance is 100s of millions of instructions per second (MIPS)
  - Over 160 processor models in the OVP Library
  - Processor pipeline and cache are not modeled

How Just-In-Time (JIT) Code Morphing Simulators Work

MULTICORE DESIGN SIMPLIFIED
imperas

- Actual production binaries are used: the software does not know that it is not running on hardware
- Code translation from target processor (e.g. ARM, MIPS, …) to host x86
- Quantum based simulation
  - All processors simulate within a quantum, then peripheral models are executed, i.e.
    1) CPU0 simulates N instructions
    2) CPU1 simulates N instructions
    3) CPU2 simulates N instructions
    4) CPU3 simulates N instructions
    5) Peripheral activities are simulated
    6) Time is then advanced to the next quantum and processor simulation starts again
  - Code translations are stored in a code "dictionary" to speed simulation during the quantum
  - Each quantum is typically 500 – 10,000 instructions
    - Quantum too small: slower simulation
    - Quantum too large: too many interrupts per quantum

- Limitations
  - No timing accuracy
  - No modeling of the microarchitectural features such as the processor pipeline

Experiment Details

MULTICORE DESIGN SIMPLIFIED
imperas

- Use virtual platforms based on ARM Cortex-A9, running Linux (v3.12.0)
- SEEs are modeled as bit flips generated randomly in any available general purpose register
  - No modification of the software or OS; fault is injected by the simulator
  - Instruction count is used as a temporal reference for fault injection
- Benchmarks run are from the Rhodinia benchmark suite

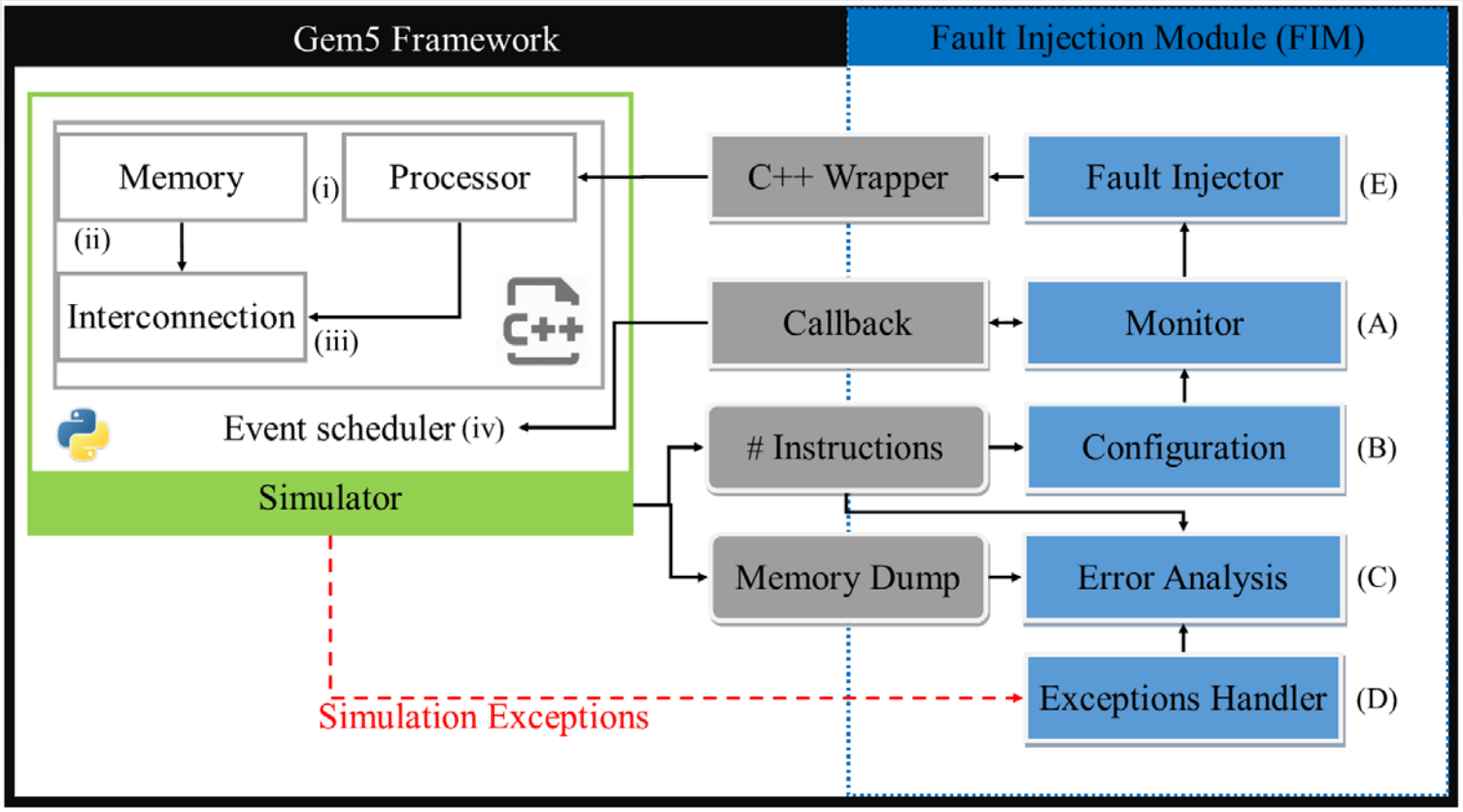
#	**Name**	**Domain**
A	bfs	Graph algorithms
B	hotspot	Physics simulation
C	hotspot3d	Physics simulation
D	Needleman-Wunsch (NW)	Bioinformatics
E	srad v1	Image processing
F	streamcluster	Data mining

Classification of Soft Errors
(according Cho et al. (DAC'13))

MULTICORE DESIGN SIMPLIFIED
imperas

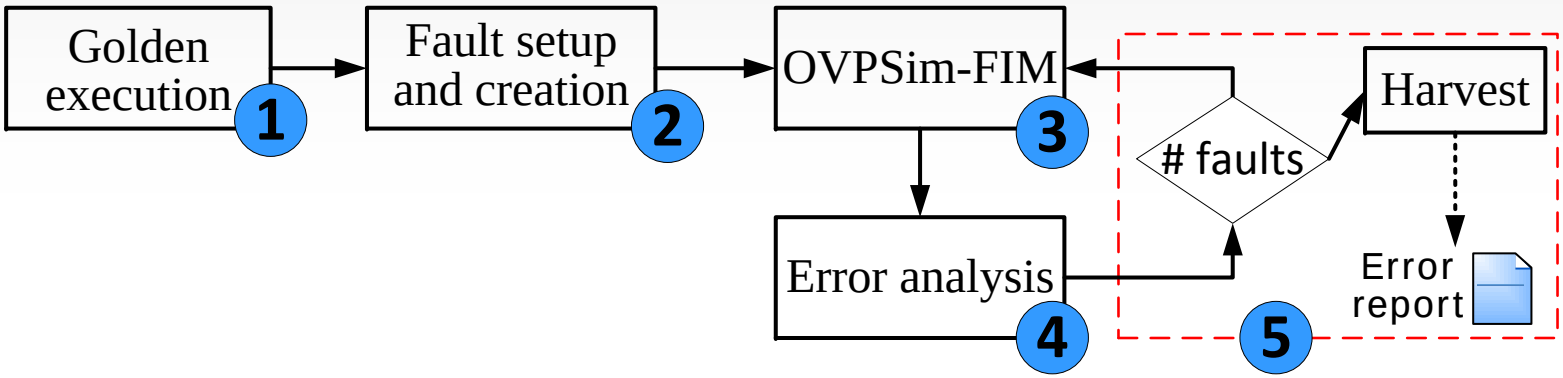
- **Vanished:** no fault traces are left
- **Application Output Not Affected (ONA):** the resulting memory is not modified, however, one or more remaining bits of the architectural state are incorrect
- **Application Output Mismatch (OMM):** the application terminates without any error indication, however, the resulting memory is affected
- **Unexpected termination (UT):** the application terminates abnormally with an error indication
- **Hang:** the application does not finish, requiring a preemptive remove

Gem5 Framework



- OVPsim framework is similar

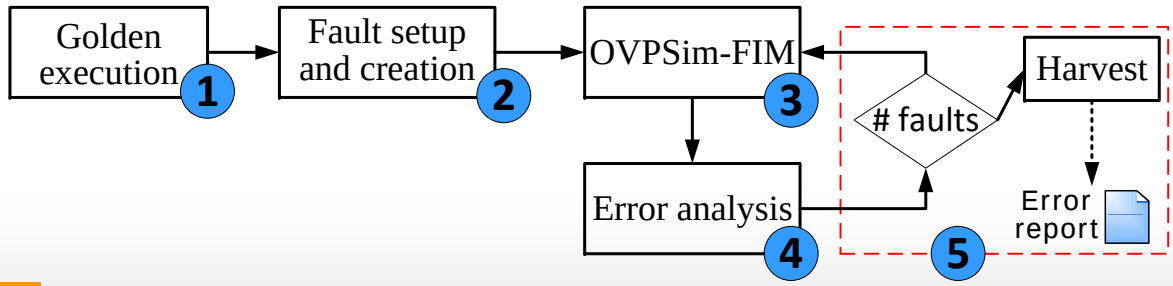
5 Phase Fault Injection Flow



- This flow is supported in gem5 and OVPsim simulators

Golden Execution

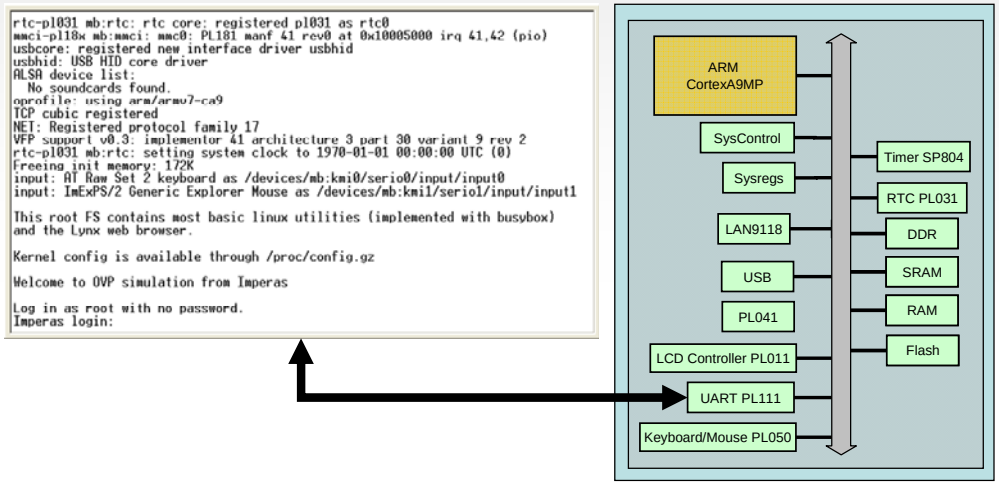
- Normal virtual platform run (fault free simulation)
  - Includes all software including operating systems, applications
- Each run captures
  - Simulation state checkpoint files
  - Memory state
  - Register values
  - Instruction counts



OVPsim Virtual Platform

MULTICORE DESIGN SIMPLIFIED
imperas

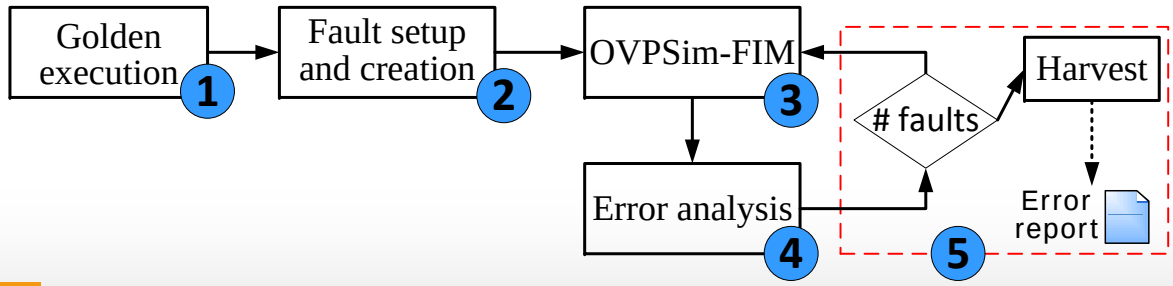
Imperas ARM platform
ARM Versatile Express Cortex-A9MP



- ARM Cortex-A9MPxN Versatile Express Platform
- Booting SMP Linux
  - Running Benchmark/Application on top

Fault Setup & Generation

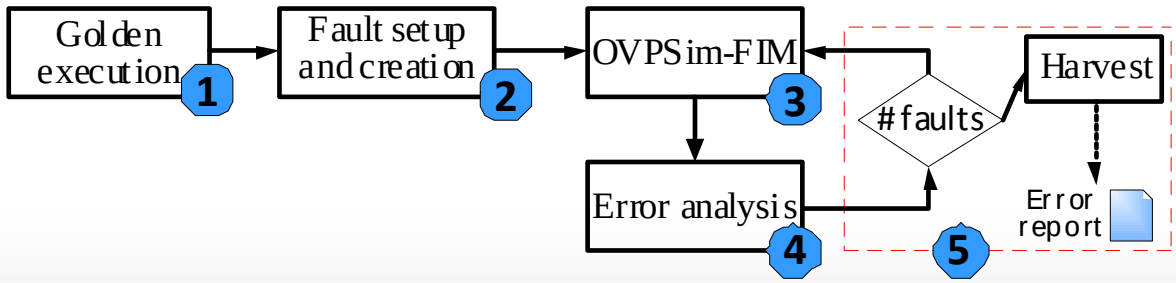
- Generate faults
  - This work: SEE (transient one time bit flip)
  - Future work: generate different faults using FIM
    - Register bit in interesting/all software functions
    - Variable values (memory locations)
    - Bus effects
- Maintain in fault dictionary for later updating and interrogation



Simulation Runs with Faults Injected

MULTICORE DESIGN SIMPLIFIED
imperas

- Inject faults:
  - 1 fault per simulation run
  - 8000 faults per campaign
- Use checkpoints (essentially starting after booting to Linux prompt) to make runs more efficient
- Classify outcomes
- Record outcome of fault simulation



Simulator – FIM Performance

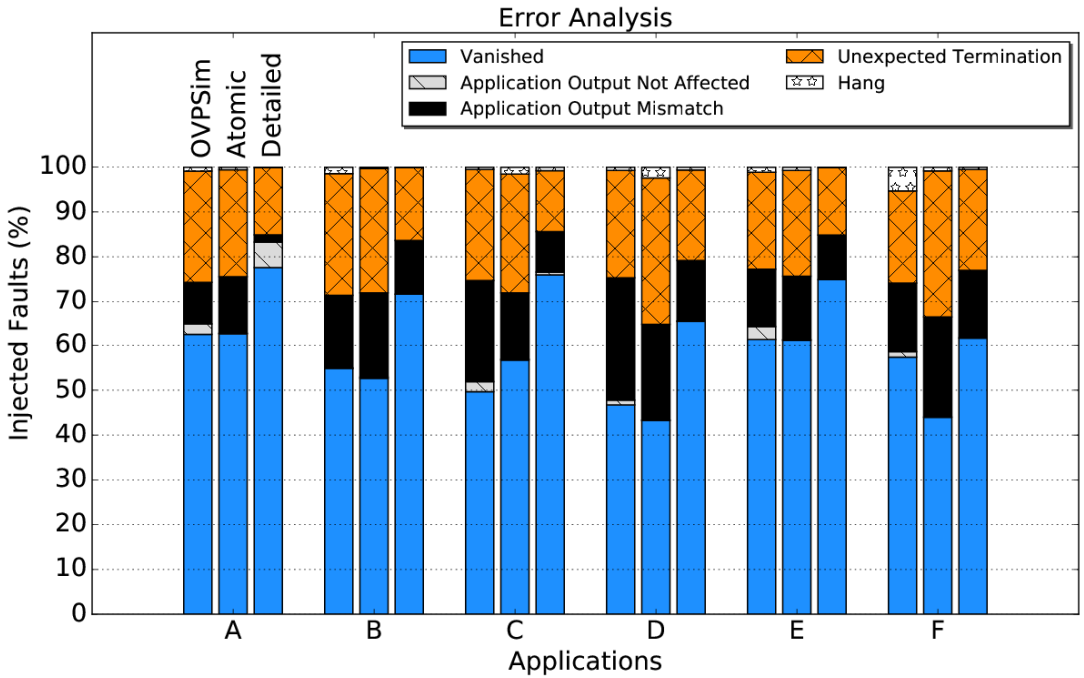
MULTICORE DESIGN SIMPLIFIED
imperas

- Example: hotspot3d benchmark with 64x64x64 matrix
- 220 million instructions
- Latest Intel i7 cores (running on HPC cluster ALICE at Univ. Leicester)

Simulator	Time for Single Simulation Run	Time for Fault Injection Campaign
gem5 Detailed	1438 sec	3600 hours
gem5 Atomic	136 sec	302 hours
OVPsim	2 sec	2.2 hours

⇒ **JIT instruction accurate simulation significantly faster than cycle accurate simulation**

Fault Injection Campaign Results

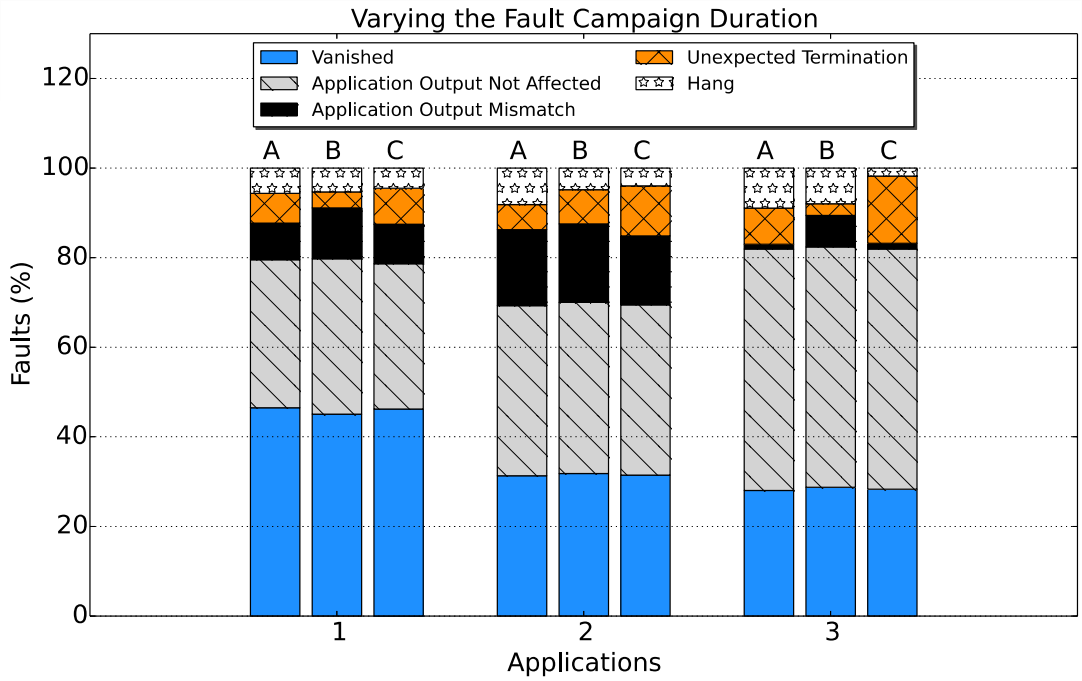


- Higher fault tolerance reported for gem5 Detailed simulations
- Unclear if this is valid, or if this is fault "masking" due to gem5 Detailed simulator artifacts (trying to mimic register renaming hardware technique)
  - Mapping could overwrite some bit-flips before next read
  - Relative similarity in results between OVPsim and gem5 Detailed for steamcluster (F) benchmark from data mining area, which has significantly higher percentage of read instructions, supports this hypothesis

Preliminary Data for Bare Metal Benchmarks

MULTICORE DESIGN SIMPLIFIED
imperas

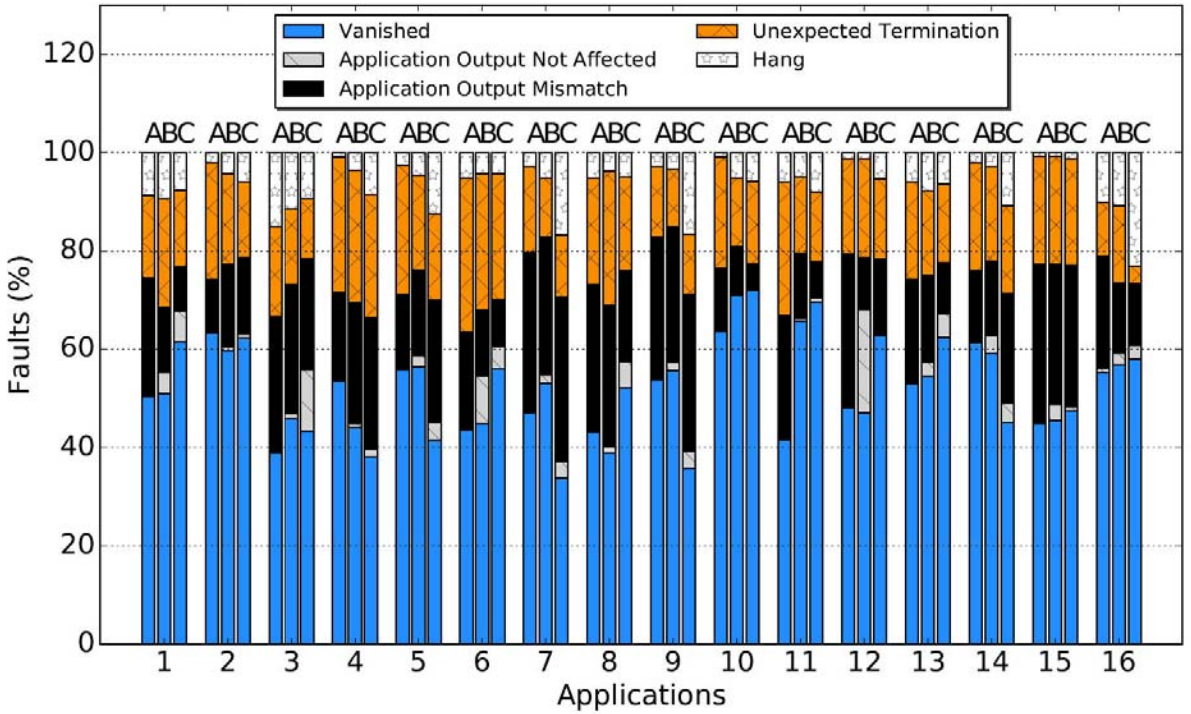
- gem5-FIM x OVP-FIM
  - A = gem5 atomic
  - B = gem5 detailed
  - C = OVPSim-FIM
- Bare metal application
  - 1 = 1 Queen (1.7 million instructions)
  - 2 = Adpcm
  - 3 = Binary Search



- Small deviation in fault detection error between Cycle Accurate (gem5) and Instruction Accurate (Imperas) simulations (A vs. C, B vs. C)

Preliminary Data: Instruction Accurate Fault Injection Results vs Number of Cores in Processor

MULTICORE DESIGN SIMPLIFIED
imperas

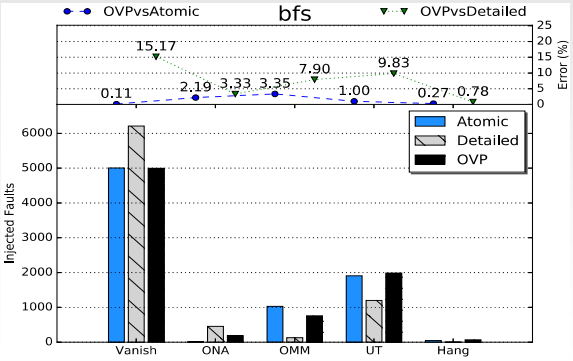


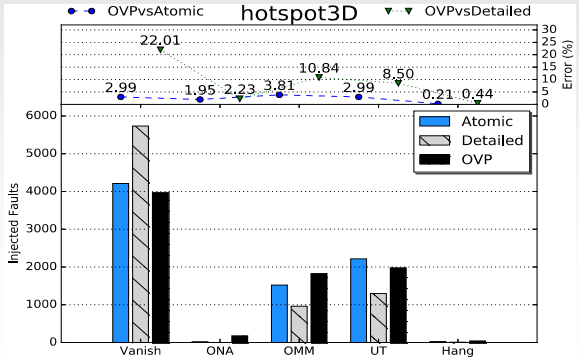
- [A, B, C] = ARM Cortex-A9MP[x1, x2, x4]
- 8,000 faults generated for each processor configuration
- 16 applications (Rhodinia HPC benchmarks) running under Linux 3.13

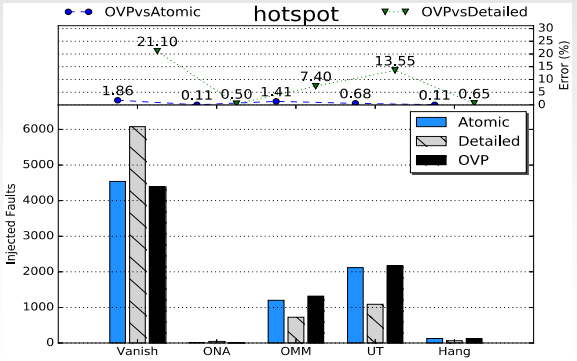
- More fault masking as number of processor cores increased? Due to simulator artifacts?

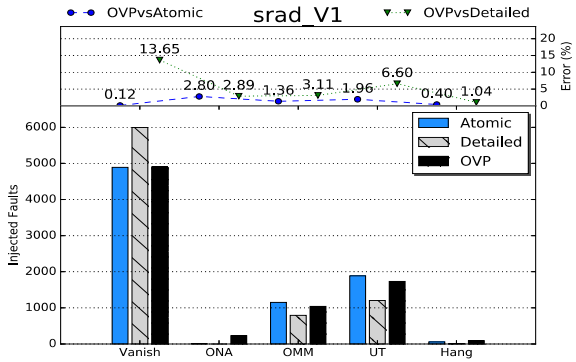
Difference Graphs for Fault Injection Categories

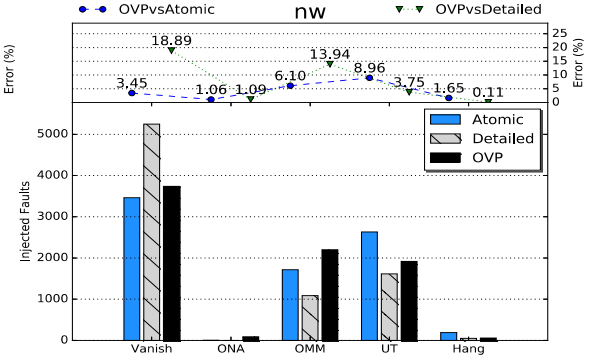
MULTICORE DESIGN SIMPLIFIED
imperas

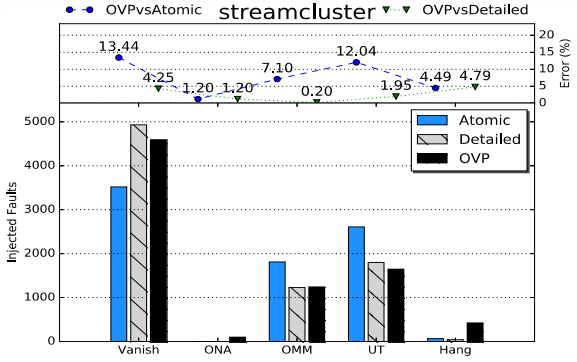












- OVPsim has similar results to gem5 Atomic
  - Therefore use OVPsim for fault injection testing since much faster
- OVPsim and gem5 Detailed results differ by 4-22%
  - Unclear if differences are due to lack of simulation accuracy with OVPsim or gem5 Detailed simulation artifacts masking real faults
  - Differences are not so significant for most fault injection testing
  - Simulation performance favors using OVPsim
  - ***Note that OVPsim Vanishing results are always less than gem5 Detailed Vanishing results, so that OVPsim is never overestimating fault tolerance***

Conclusions

MULTICORE DESIGN SIMPLIFIED
imperas

- JIT-based instruction accurate simulation was compared to cycle accurate simulation for fault injection testing
- Small, acceptable error compared to cycle accurate fault simulation
- Significant performance advantage for instruction accurate over cycle accurate simulation with fault injection

- Instruction accurate simulation with fault injection can be used for
  - SEE analysis at early design phases
  - Comparisons of different processor architectures, software architectures

- Future work
  - Increasing fault coverage
  - Understanding and reducing simulator artifacts