GemFI: A Fault Injection Tool for Studying the Behavior of Applications on Unreliable Substrates

CERTH
The Centre for Research & Technology Hellas
Department of Electrical & Computer Engineering, University of Thessaly, Greece
Northwestern University

KONSTANTINOS PARASYRIS, GEORGE TZIANTZOULIS,
CHRISTOS D. ANTONOPOULOS, NIKOLAOS BELLAS
koparasy@inf.uth.gr, georgiostziantzioulis2011@u.northwestern.edu, cda@inf.uth.gr, nbellas@inf.uth.gr

1. Motivation

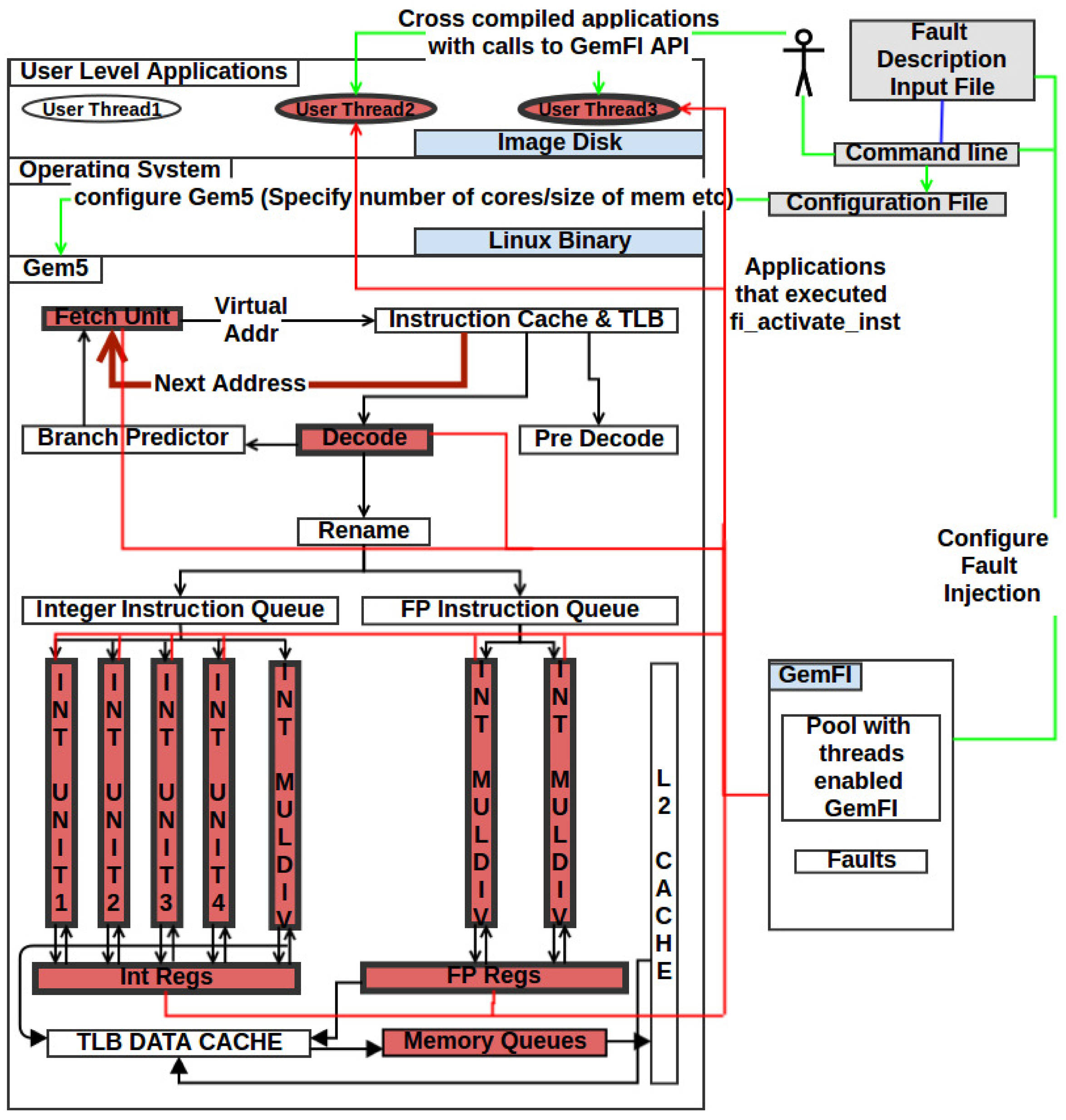
- Power consumption remains #1 constraint of future systems.
- **Reducing supply voltage** below nominal values results to significantly lower power consumption at the expense of **potential errors**.
- Several application domains offer the opportunity to **trade-off quality of service** for significant **improvements in power/energy consumption.**
- Reliable computing under unreliable circumstances is the next challenge the computing community must solve.
- There is a need to perform a thorough analysis of the way hardware faults manifest errors to architectural components and how errors affect the applications' behavior.

2. Objectives

- Create a full system fault injection and analysis tool.
- Easily extensible to cover various ISAs and CPU configurations.
- Support **multiple fault models**.
- **Mitigate the simulation time overhead**.

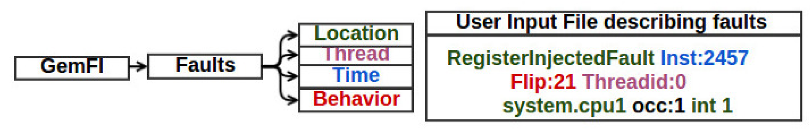
3. Our Tool

- GemFI is based on the **Gem5 simulator**.
- Allows fault injection in both functional and cycle-accurate simulations.

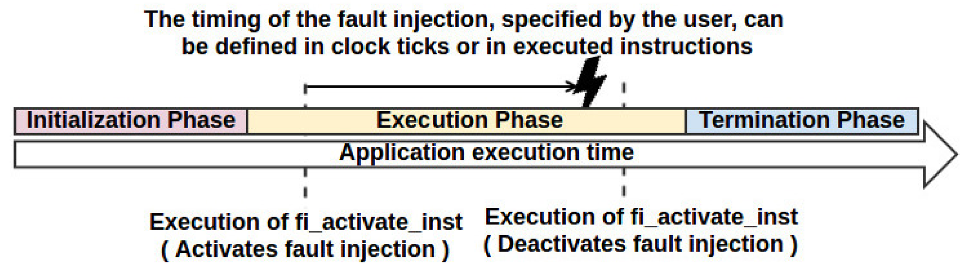


Red components demonstrate possible fault locations, red ovals represent fault injected applications.

- Faults are described in an input file provided by the user.



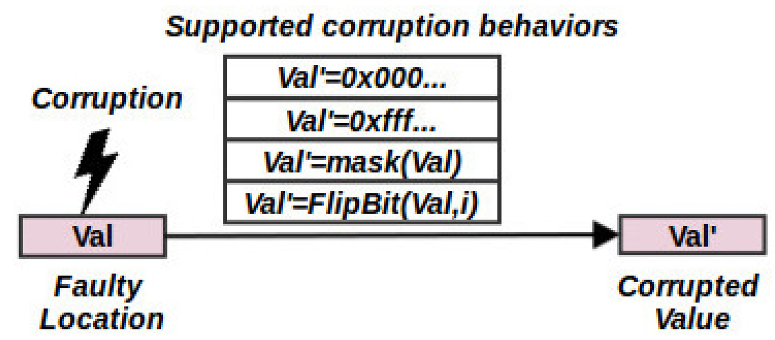
- The time of a fault specifies the timing of fault manifestation.



- Provides function calls to permit fault injection to specific applications/threads.

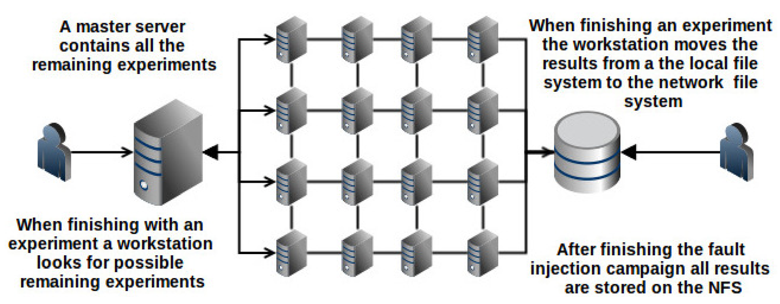
```
#include <m5op.h>
int main(int arg, char *argv[]){
  int id = 0;
  initialize_input_data();
  fi_read_init_all();
  fi_activate_inst(id);
  foo();
  fi_activate_inst();
}
```

- Values in a location can be corrupted in a variety of ways.

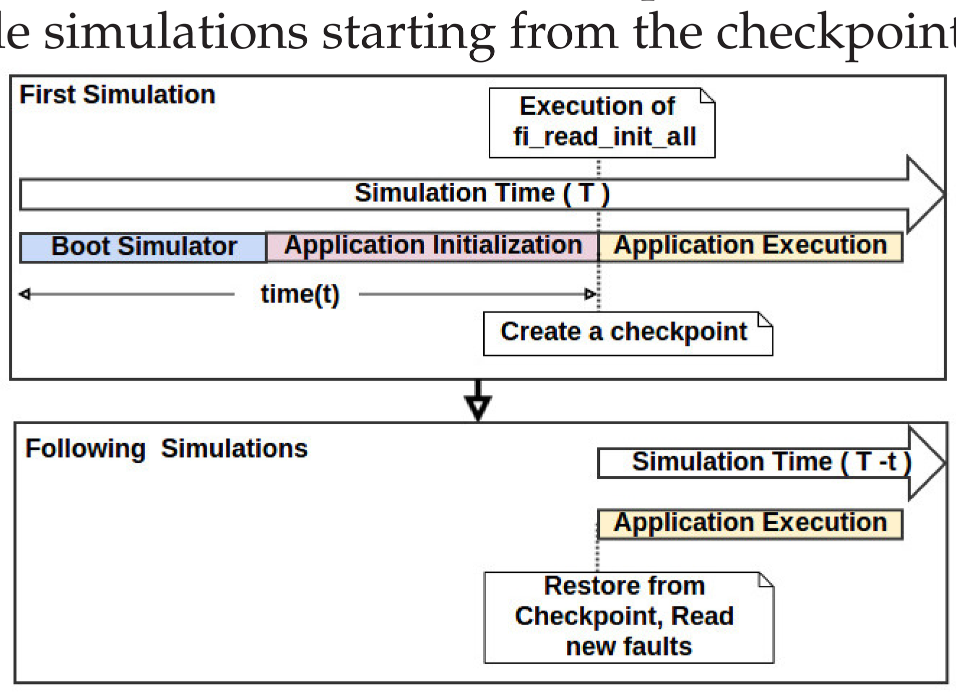


4. Optimization Techniques

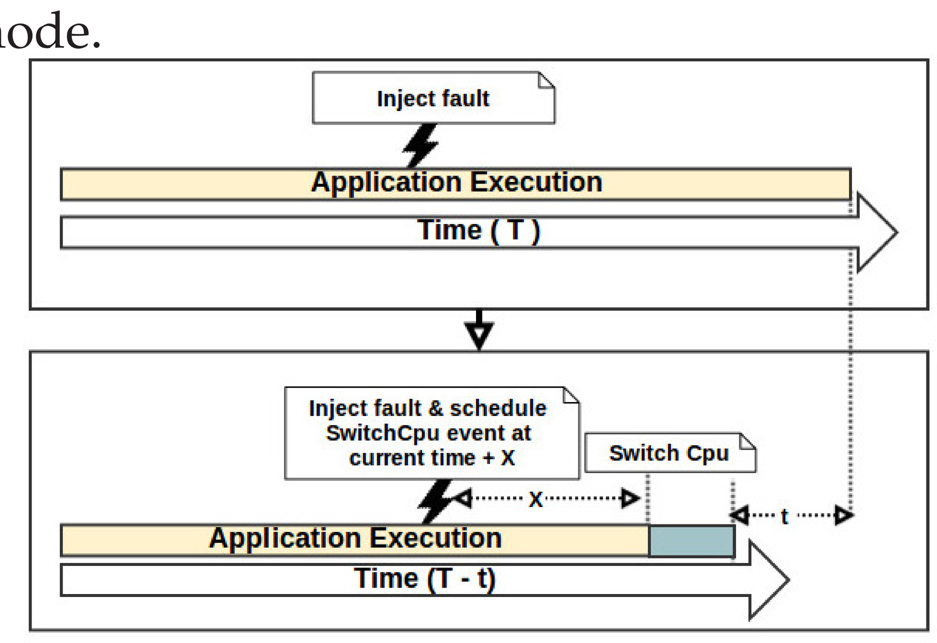
- Entire **simulation can be executed in parallel**.



- Each workstation may execute more than one experiment simultaneously, depending on the number of cores and RAM configuration.
- Checkpointing is necessary to avoid loss of simulations.
- *"Clever"* **checkpointing can also speed-up simulations.**
- Execute once up to the point of boot up & application initialization and checkpoint. Start multiple simulations starting from the checkpoint

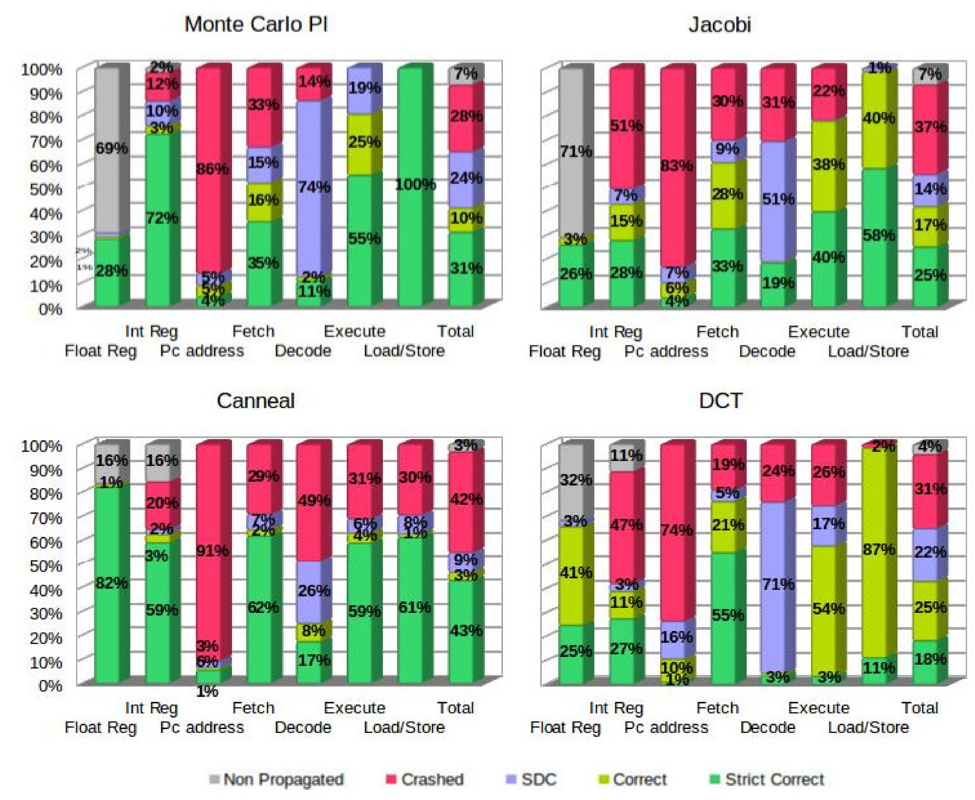


- Inject a fault and wait until the fault manifests or is masked, then switch to a faster simulation mode.

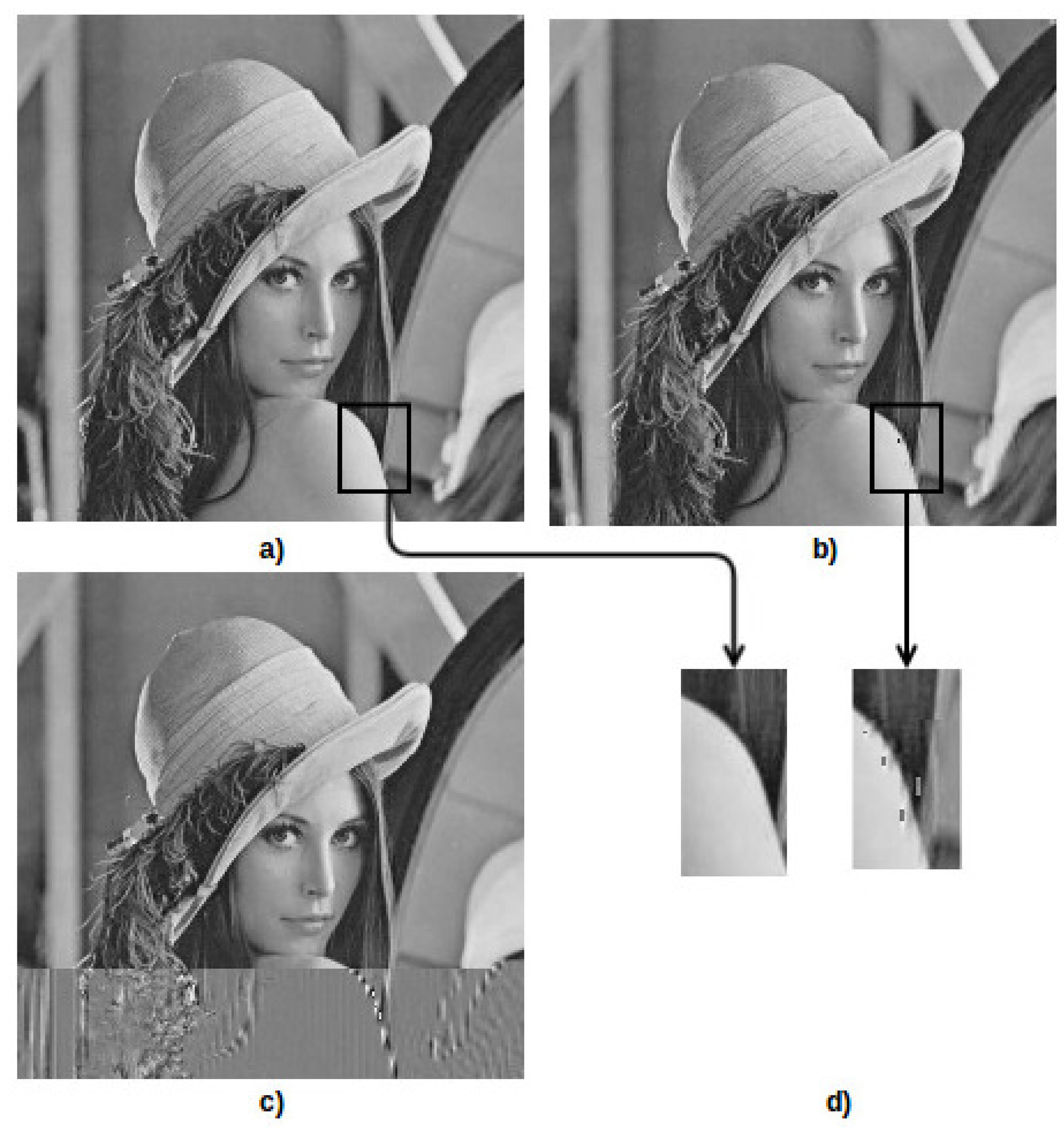


5. Results

- Running simulations in parallel in **27 workstations with 4 cores per workstation we obtained speedups of 103x**.
- The **checkpointing** methodology results to additional **speedups varying from 2.36x up to 41.84x**.
- We executed **2500 fault injection campaigns** per application.

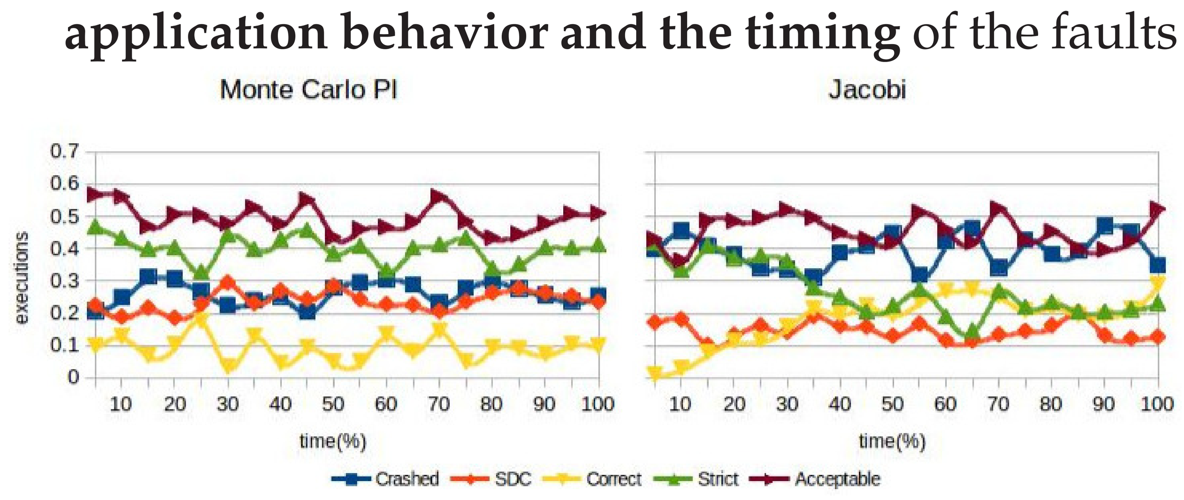


- Results are categorized as:
  - *Strictly correct*: bit-wise identical results in comparison with an error-less execution.
  - *Correct*: results not strictly correct, however still within acceptable quality margins.
  - *SDCs*: experiments terminate normally, yet the output quality is not acceptable.
  - *Crashed*: experiments fail to terminate.
  - *Non propagated*: faults did not manifest as errors.
- Tolerance to injected faults proved **highly dependent on the targeted hardware module**.
  - PC and IFetch modules are very vulnerable even to single errors.
  - Arithmetic operations are often error tolerant, if they are not used for address calculations.



Different categories of results for the DCT benchmark. a) A strict correct result b) Relaxed correct result c) SDC d) The difference between (a),(b) (loss of quality)

- In some cases we observed a **correlation between application behavior and the timing** of the faults



- Tolerance to injected faults is **dependent on the inherent characteristics of each application**.
  - Applications with intensive memory address calculations often result to segmentation faults (for example Canneal) whereas computationally heavy applications (like *Monte carlo estimation of PI*) are characterized by lower crash rates.

This work has been partially supported by the EC within the 7th Framework Program under the FET-Open grant agreement SCoRPiO, grant number 323872.