FIMSIM: A Fault Injection Infrastructure for Microarchitectural Simulators

Gulay Yalcin*, Osman S. Unsal*, Adrian Cristal* †, Mateo Valero*
* Barcelona Supercomputing Center
† IIIA - Artificial Intelligence Research Institute - CSIC - Spanish National Research Council
Email: (gulay.yalcin, osman.unsal, adrian.cristal, mateo.valero)@bsc.es

Abstract—**Fault injection is a widely used approach for experiment-based dependability evaluation. Injecting faults to microarchitectural simulators is particularly appealing for researchers, since it can be utilized at the early design stage of the processor. As such, it enables a preliminary analysis of the correlation between the criticality of processor-structure level faults and their impact on applications. In this study, we present FIMSIM, a compact fault injection infrastructure for microarchitectural simulators which is capable of injecting transient, permanent, intermittent and multi-bit faults. FIMSIM provides the opportunity to comprehensively evaluate the vulnerability of different microarchitectural structures against different fault models.**

I. Introduction

Providing resilience against hardware faults (transient/intermittent/permanent) has become a first class design constraint. It is also essential to carefully evaluate the level of dependability that a fault tolerance approach provides besides its performance impact. Fault injection is a widely used experiment-based dependability evaluation approach in which faults are injected either (1) to the real hardware, (2) to an architectural simulator or (3) to the system or application software. Simulation based fault injection is more appealing for researchers since it is applicable early in the design time and it can inject faults to the processor structures that cannot be exercised by injecting faults at the software level. Several simulation based fault injectors are implemented at Register Transfer Level (RTL) [2], [4] which are incapable of modeling a wide range of systems due to their design complexity and their high simulation time. Moreover, they are impractical for the researchers who develop resilience mechanisms at the microarchitectural level.

In this study, we present FIMSIM, a fault injection infrastructure for microarchitectural simulators. FIMSIM is capable of injecting transient, permanent and intermittent faults either in isolation or as a combination of these fault models. In addition, it can inject multi-bit faults. Hence, FIMSIM is convenient for the evaluation of dependable systems. Moreover, FIMSIM, unlike prior fault injectors, injects faults to critical small sized structures (bypass logic, PC) besides other large buffered structures since faults in these small-sized but vulnerable structures may lead drastic errors. Inevitably, the sizes of structures are taken into account by FIMSIM for the calculation of the vulnerability of the entire microarchitecture. Consequently, the compact characteristic of FIMSIM provides opportunity to make a comprehensive evaluation of the vulnerability of different microarchitectural structures against different fault models. Besides determining the performance impact in the fault free cases and calculating the dependability level that a fault tolerance scheme presents, recovery time is also a crucial constraint for fault tolerance which has not been integrated to previous fault tolerance simulators. FIMSIM makes it easy to integrate the recovery time measurement experiments.

II. Implementation

We enhance M5 [1], a popular open-source microarchitecture simulator with a large user-base, by fault injection capability. M5 is fairly convenient for a fault injection tool since it is a deterministic full system simulator and it provides a checkpointing mechanism that dumps the whole inner-state of the architecture to a checkpoint file.

In FIMSIM, the user defines the list of fault(s) in the input file by setting the following properties of fault(s): (1) *processor id*, (2) *fault type* (transient, stuck-at-0, stuck-at-1, dominant-0, dominant-1, intermittent-0, intermittent-1), (3) *fast-forward cycle*, (4) *fault injection cycle*, (5) *the faulty structure* (intRF, specialRF, ALU, ITB, DTB, Bypass, PC), (6) *the faulty entry* (e.g. register number), (7) *the faulty bit number*, (8) *persisting time* of the intermittent faults, (9) *neighbor fault id* for multi-bit fault injections, (10) *direction* of neighborhood for spatial faults (vertical, horizontal). These parameters explicitly define a fault so that the user can repeat a fault injection experiment for debugging the effect of a particular fault definition. Note that, the user can prefer any/all of these parameters to be random so that multiple fault injection campaigns could be conducted and their results processed to calculate overall processor reliability. At the beginning of a fault injection simulation, FIMSIM reads the list of faults from the input file. During the simulation, at every simulation cycle, M5 calls the fault injection method which first compares the injection cycle of each fault and the simulation cycle. If the simulation cycle equals injection cycle, FIMSIM modifies the corresponding value in the pertinent structure according to the fault type of the fault. If the injected fault is transient, it is deleted from the fault list after the first injection. Otherwise (permanent and intermittent), it keeps being injected in the following cycles either until the end of the simulation (permanent) or until

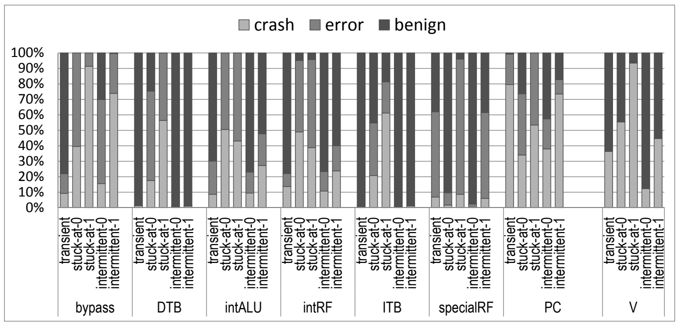

Fig. 1. Single Fault Injection to Spec2006 Benchmark.

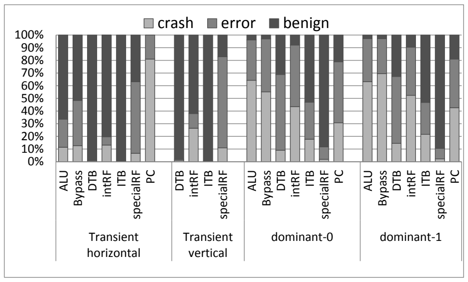

Fig. 2. Multi-bit Fault Injection to spec2006 benchmark.

the persisting time of the fault (intermittent). At the end of each fault injection simulation, a checkpoint is created in the following checkpoint creation cycle unless the fault causes the application to crash. In order to produce an error-free checkpoint, FIMSIM is executed once for golden run. Each checkpoint produced after a fault injection is compared with the one produced after golden run in order to see whether the fault has completely disappeared (masked) or it still stays in the architecture. FIMSIM classifies the faults into three groups according to the way they manifest themselves: (1) benign faults, (2) system crashes and (3) errors. Finally, when calculating the vulnerability of whole architecture, the error rate of each structure is multiplied by the likelihood of the fault occurrence in the structure (e.g. size of the structure). The technical details of the vulnerability calculation of the entire processor is explained in [6].

III. Evaluation

In this section, we evaluate the vulnerability of in-order Alpha 21264 microarchitecture by utilizing spec cpu2006 benchmark suite. We inject the faults to five different structures in the core; bypass logic, data TLB (DTB), instruction TLB (ITB), arithmetic logic unit (ALU), integer register file (int-RF), special purposed register file (RF-special) and program counter (PC). Note that, in-order cores do not have some complex structures (e.g. ROB) required for out-of-order execution. We inject 100 faults per structure in each application to a random location in the structure (i.e. 1400 faults/structure). which is similar to or considerably higher than prior fault injection analyses [3], [5]. We generate checkpoints at every 100M cycles after warming up 70M cycles. Figure 1 presents the single fault injection results. Unsurprisingly, processors are most vulnerable to permanent faults. The interesting result is that stuck-at-1 faults are more harmful for the applications than stuck-at-0 faults. This is because bit values are mostly zero (e.g. more than 70% of bits in PC and more than 90% of bits in special register file are zero). Thus, stuck-at-0 faults (permanent or intermittent) are generally benign. When we compare the effects of the faults on ALU and bypass logic, the bypass logic is slightly more vulnerable to the faults. This is because the fault affects the next instruction in the bypass logic. In TLBs (ITB or DTB), short term faults (transient or intermittent) are compensated. However, when there is a permanent fault in TLB, it is more detrimental for processor reliability. The PC is the most vulnerable structure in that any fault in the PC results in either an error or system crash. For multi-bit faults (Figure 2), the multi-bit transient faults in the horizontal direction are not more harmful than a single bit transient fault. However, in the vertical direction, multi-bit transient faults become significantly more harmful in the register files since it affects more than one entry in the buffer. Dominant-0 and dominant-1 faults affect the vulnerability in the similar way since the final result of the bit values changes if two bits are different from each other meaning they are effective in the same conditions. In Figure 3 we present the recovery time of SymptomTM [7], a transactional memory based fault tolerance schema, as a case study. The recovery time of an error is based on the transaction size in SymptomTM. For instance, if a fault affects a large transaction, the recovery time increases (e.g. astar).

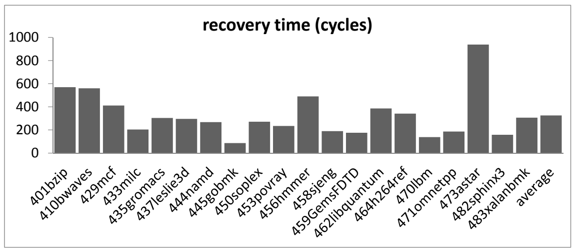

Fig. 3. Recovery time of SymptomTM, a fault tolerance schema based on transactional memory

References

[1] N. L. Binkert and et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26:52–60, July 2006.

[2] D. Gil and et al. Study, Comparison and Application of Different VHDL-based Fault Injection Techniques for the Experimental Validation of a Fault-Tolerant System. *Microelectronics Journal*, 34(1):41–51, 2003.

[3] M. lap Li and et al. Accurate Microarchitecture-Level Fault Modeling for Studying Hardware Faults. In *HPCA*, 2009.

[4] N. J. Wang and et al. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *DSN*, pages 61–70, 2004.

[5] N. J. Wang and et al. Examining ACE Analysis Reliability Estimates Using Fault-Injection. In *ISCA*, pages 460–469, 2007.

[6] G. Yalcin and et al. FIMSIM: Technical Report, https://gw.ac.upc.edu/go/intranet-1/cgi-bin/apps/reports/entry_point.pl, 2011.

[7] G. Yalcin and et al. SymptomTM: Symptom-Based Error Detection and Recovery Using Hardware Transactional Memory. In *PACT*, 2011.