FIMSIM: A Fault Injection Infrastructure for Microarchitectural Simulators

Abstract—**Fault injection is a widely used approach for experiment-based dependability evaluation in which faults can be injected to the hardware, to the simulator or to the software. Simulation based fault injection is more 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 circuit level faults and their impact on applications. However, the lack of publicly available fault injectors for microarchitecture level simulators brings extra burden of designing and implementing fault injectors to the researchers who evaluate microarchitecture dependability. In this study, we present FIMSIM, to the best of our knowledge, the first publicly available fault injection simulator at the microarchitecture level. FIMSIM is a compact tool which is capable of injecting transient, permanent, intermittent and multi-bit faults. Therefore, FIMSIM provides the opportunity to comprehensively evaluate the vulnerability of different microarchitectural structures against different fault models.**

I. Introduction

Dependability against hardware faults (transient/intermittent/permanent faults) has become a first class design constraint for computer designers due to scaling technology trends (e.g. smaller feature size). Hence, several fault tolerance techniques have been proposed to produce robust and reliable computer systems [17], [20], [21], [22], [23], [29]. However, fault tolerance techniques introduce penalties in performance, in power, in die size or in design time. Therefore, it is essential to carefully evaluate the level of dependability that the fault tolerance approach provides.

Fault injection is a widely used experiment-based dependability evaluation approach in which faults are injected either (1) to the real hardware, (2) to the simulator or (3) to the software (operating system or application). While hardware fault injection requires at least the physical prototype of the system, software fault injection is limited in the sense that they cannot inject faults into locations that are inaccessible to the software. On the other hand, simulation based fault injection is applicable early in the design time and it can inject faults to the processor structures that cannot be excited by injecting faults at the software level. Nevertheless, it provides the correlation between the criticality of circuit level faults and their impact on the application level. Consequently, simulation based fault injection is more appealing for researchers than software and hardware based fault injections.

Several simulation based fault injectors model the Register Transfer Level (RTL) of microprocessors [9], [27]. These models are incapable of modelling a wide range of systems (e.g. multi core multi threaded architectures) due to their design complexity and their high simulation time. Moreover, they are impractical for the researchers who design fault tolerance in the microarchitecture level since they use microarchitectural simulators for their design. However, the lack of publicly available microarchitecture level fault injector simulators leads each researcher to implement his/her own fault injector [14], [26] to test fault tolerant schemes. However, this brings extra design and implementation burden to the researchers.

In this study, we present FIMSIM, to the best of our knowledge, the first publicly available fault injection infrastructure for microarchitectural simulators. Therefore FIMSIM takes the burden of designing and implementing a fault injection tool from the researchers.

Fault tolerant schemes generally targets particular fault models. FIMSIM is capable of injecting transient, permanent and intermittent faults either as single-bit fault or multi-bit faults as combination of several fault models. Hence, FIMSIM is convenient for the evaluation of many dependable system. On the other hand, faults in small-sized structures (bypass logic, PC) may lead to drastic errors despite the fact that the likelihood of the fault occurrence in a bigger-sized structure (e.g. register file) is higher. FIMSIM, apart from the prior fault injectors, injects faults to critical small sized structures besides other large buffered structures. Inevitably, the sizes of structures are taken into account by FIMSIM for the calculation of the dependability 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. In this work, we present the detailed analysis of the vulnerability of the in-order Alpha [1] microarchitecture by utilizing spec cpu2006 [11] benchmark suit.

Besides performance impacts in the fault free cases and dependability level that a fault tolerance scheme presents, recovery time is also a crucial constraint for fault tolerance which has not been measured by many prior fault tolerance proposal. FIMSIM supplies recovery time measurement as well as dependability evaluation. In this study we evaluate the recovery time of FaulTM [29], a transactional memory based fault tolerance technique by utilizing FIMSIM, as a case study.

Simulation time is a prominent limitation for the fault injection technique. FIMSIM, reduces the simulation time due to two reasons. First, FIMSIM is on microarchitecture level simulation which keeps the simulation time shorter compared to RTL level simulations. Second, we utilize M5 [4], a microarchitecture simulator that provides a checkpointing mechanism for the implementation of FIMSIM. Therefore, numerous fault injections, especially in the late phase of the applications, are achieved in shorter amount of time by restoring the checkpoints in the later phase without the need to run the application from the beginning.

Repeatability of fault injection experiments (injecting the exact fault again) is also an essential issue for validation of dependability since, compelling test cases can be determined and prepared. FIMSIM is able to repeat a fault injection experiment easily. In FIMSIM, the user can define the fault injection point explicitly instead of injecting faults to randomly generated points.

The contributions of this study are:

- We present FIMSIM, a publicly-available microarchitectural simulation fault injection infrastructure.
- FIMSIM is a compact tool which is capable of injecting transient, permanent, intermittent and multi-bit faults.
- We present a comprehensive evaluation of the vulnerability of different microarchitectural structures in Alpha microarchitecture against different fault models.
- Through a case study, we demonstrate how FIMSIM facilitate detailed reliability analyses. In particular, we examine the fault recovery time overheads of the FaulTM [29] fault tolerance mechanism.

In the next section, we present the fault model that FIMSIM supports. In Section III, we explain the design principles of FIMSIM. In Section IV, we present our fault injection result. We will discuss the prior studies in Section V.

II. Fault Model

Faults experienced by semiconductor devices fall into three main categories: transient, intermittent and permanent. Moreover, when these faults affect more than a bit at a time, multi-bit faults occur. In this section we will explain these faults.

A. Transient Faults

A *transient fault* (also known as Soft Error) is a bit flip due to some radiation event or power supply noise. Since the data bit stored in a device is corrupted until new data is written to that device, these errors are temporal (transient) [2]. Despite the fact that transient faults are nondestructive functional errors and they can be fixed by re-setting or re-writing of the device, they may cause dramatic impact on computer systems unless they are mitigated [3]. As transistor dimensions and operating voltages shrink, sensitivity to radiation increases dramatically. Thus, it is foreseen that future systems will be more prone to transient faults. In FIMSIM, we simulate transient faults by flipping (changing 0 to 1 or vice versa) the value of the randomally selected bit in a randomly selected cycle.

B. Permanent Faults

Irreversible physical changes in the semiconductor devices are called *permanent faults*. Fault tolerance mechanisms usually disconnect the faulty structures hit by permanent faults, and replace them with fault-free spare structures. Systems having these mechanisms tolerate permanent faults well. In fact, the lifetime reliability of a system is defined by its ability to tolerate these faults. Permanent faults tend to occur early in the processor lifetime due to manufacturing faults (called “infant mortality”), or late in the lifetime due to thermal and process related stress (typically faults in this epoch are manifested first as intermittent faults, then progress to permanent faults). In FIMSIM, we use stuck-at-0 and stuck-at-1 fault models to simulate the permanent faults in which a randomly selected value becomes stuck from the fault injection cycle until simulation ends.

C. Intermittent Faults

Process variation or in-progress wear-out, combined with voltage and temperature fluctuations cause burst of frequent faults, called *intermittent faults*, that last from several cycles to several seconds. An intermittent fault occurs repeatedly at the same location, it tends to occur in bursts for a period of time when the fault is activated while replacement of the offending circuit removes the intermittent fault [6], [28]. It has been suggested that intermittent faults have the potential to impact program execution to a greater extent when compared with transient faults [19]. Moreover, it is hard to diagnose by post facto using hardware/software tests because intermittent faults do not persist and the conditions that caused the fault are hard to regenerate.

Continued device scaling results in increased Process, Voltage and Temperature (PVT) variations, increased cross-talk and decreased noise margins all of which lead to increased susceptibility to intermittent faults. Moreover, wear-out failures are expected to become much more frequent but devices typically do not fail suddenly, they display intermittent behaviour for a period of time beforehand. Therefore, it is prevised that the rate of occurrence of the intermittent faults will increase [5], [7].

In FIMSIM, we simulate intermittent faults by utilizing stuck-at fault models for a predefined number of cycles.

D. Multi-bit Faults

Multi-bit faults occur when hardware faults affect multiple bits at a time. In this section we will explain several of them.

Spatial multi-bit upsets of transient faults occur when a single particle strike causes more than one bit-flip in the neighbourhood. Results of irradiation tests on 90nm commercial processes reveal that multi-bit upsets as large as 13 bits can occur in sub-100nm technologies [18]. These faults are expected to increase in the future processors due to shrinking size of the transistors. A two-bit spatial multi-bit upset can manifest in two ways: horizontal or vertical [8]. Horizontal means two adjacent bits on the same word are upset. Vertical, on the other hand, means that the same bit in two adjacent words are upset.

Temporal multi-bit upsets happen when multiple independent particles strike distinct locations of the structure causing upsets on multiple bits. Since the likelihood of particle strike is high in the high altitudes, the probability of temporal multi-bit upsets increases in higher altitudes [16].

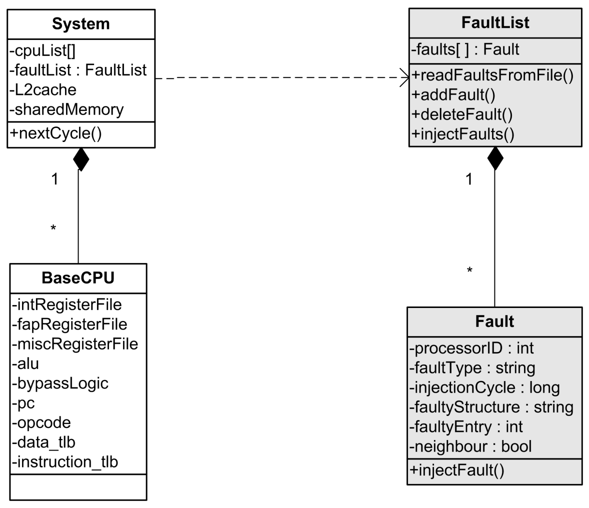

Fig. 1. Class Diagram of FIMSIM. Shaded classes are the extensions for the fault injection purpose.

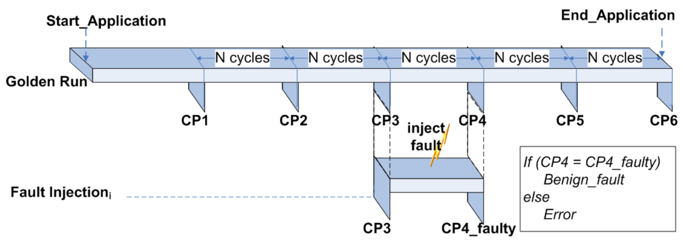

Fig. 2. Checkpoint generation in golden run

Bridging permanent faults are caused by shorts between normally unconnected signal lines. There are two types of bridging faults (1) dominant-1 is modelled assuming that there is an AND gate between bits and (2) dominant-0 is modelled by using an OR-gate. Note that, if there is a short between two bits, only the value of one bit changes. However, we classify these bridging faults under multiple faults since it is related to multiple bits.

III. Implementation

In this section, we describe the infrastructure of the FIMSIM tool and its capabilities. First, we define the simulator that we enhanced with fault injection capability. After that, we describe the aspects of a golden run. Then, we explain the fault injection implementation. Finally, we present the calculation of the processor dependability according to the fault injection results.

A. Simulator

We enhance the M5 full system simulator [4] with fault injection capability. We select M5 for our base simulator due to its several properties that are convenient for a fault injection tool. First, M5 is a full system simulator that we can observe the effects of hardware faults on the operating system and user applications. Second, M5 has a checkpointing mechanism that dump the whole inner-state of the architecture to a checkpoint file whenever it is desired. In FIMSIM, we can trace the effect of a fault by comparing checkpoints easily. Moreover, we can accelerate repetitive fault injections to the late phase in applications by restoring the checkpoints without having to execute until that point. Note that M5 is a deterministic simulator that the results do not vary at all between two identical runs at different times. Third, in M5, we can add command line options easily by modifying Python scripts. This is beneficial because we can define the fault injection and the golden run options without modifying the simulation fundamentals. Fourth, the M5 is implemented in an object oriented programming manner so that we can enhance the simulator with fault injection capabilities without modifying the rest of the simulation dramatically. Fifth, M5 is a popular open-source simulator with a large user-base, therefore it is a good substrate for FIMSIM; especially with a view towards releasing FIMSIM to the reliability research community.

Figure 1 presents the fault injection classes that we added to the M5 simulator and how these fault injection classes interact with the existing classes in M5. The white area was already implemented by M5, we add the shaded classes: *FaultList* and *Fault*.

B. Golden Run

In the golden run which is executed only once for each application, FIMSIM produces error free checkpoints periodically during the execution of the entire application. For example, in Figure 2, FIMSIM generates checkpoints at every N cycles (e.g. N = 10M), so that 6 checkpoints are generated for the whole application. For instance, to inject faults to the 4th chunk of the application, CP3 is loaded to the FIMSIM simulator without having to execute the application from the beginning. After the fault injection, CP4 of the golden run and the faulty run are compared without waiting until the end of the application in order to see whether the fault has completely disappeared (masked) or it still stays in the architecture. We argue that instead of comparing the final results of the application, comparing the architectural states is essential, because a fault may stay in the system or worse propagate to the operating system although it does not affect the final output of the application.

C. Fault Injection

In FIMSIM, the user defines the list of fault(s) in the input file (e.g. a single permanent fault and/or spatial multi-bit transient faults). In this file, the user defines each fault with the following properties:

- the processor id where the fault will be injected (0..(number of processor-1)),
- fault type (transient, stuckat0, stuckat1, dominant0, dominant1, intermittent0, intermittent1),
- fast-forward cycle before fault injection
- the cycle of the fault injection (0..(total Execution Cycle -1)),
- the faulty structure (intRF, specialRF, ALU, ITB, DTB, Bypass, PC),

- the faulty entry (e.g. register number),
- the faulty bit number (0..31 for 32 bit machine).
- persisting time of the intermittent faults,
- neighbour fault id for multi-bit fault injections (0..(number of faults-1)).
- direction of neighbourhood 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 the simulation, FIMSIM reads the list of faults from an input file and it sets the list defined by the `FaultList` data structure. Making a list of faults gives an opportunity to add multiple bit faults at a time. Some of these multi-bit faults affect the neighbour bits as well. In FIMSIM, these faults are described with the neighbour attribute to define the place of the second bit. For example, a spatial multi bit fault is defined as two faults that are neighbours to each other. In this case, the second fault takes the place of the fault from the first fault's options and it is located just next to it.

After starting the simulation, at every cycle (e.g. in `nextCycle()` method), M5 calls the `FaultList`'s `injectFaults()` method which also calls the `injectFault()` method of each fault. `injectFault()` method first compares the injection cycle of the fault and the executing cycle of the simulator. If the executing cycle is late enough, FIMSIM modifies the corresponding value in the pertinent structure according to the fault type of the fault. For instance, for a transient fault injection to the register file, FIMSIM flips the corresponding bit in the register file. After the injection, if the injected fault is transient, it is deleted from the fault list. Otherwise (permanent and intermittent), it keeps being injected in the following cycles either until the end of the simulation (permanent) or until the persisting time of the fault (intermittent).

In each fault injection, the checkpoint is created in the following checkpoint creation cycle unless the fault causes the application to crash. This faulty checkpoint is compared with the one produced in the golden run to see whether the fault is masked or it has changed the state of the microarchitecture. Thus, there are three possible outputs of the fault injection:

- **Crash:** The application crashes before the simulation reaches to the checkpoint creation cycle therefore it does not generate a checkpoint. This crash might be due to a segmentation fault or a fatal trap exception (e.g. incorrect program counter).
- **Benign:** If the checkpoints of the faulty run and the golden run are identical, that means that the injected fault was masked (e.g. a faulty register is written before it is read by any instruction)
- **Error:** If the faulty checkpoint is different from the golden one, that means that the fault led to a different architectural state. This error group also includes the silent data corruptions (SDC).

D. Processor Dependability

When calculating the vulnerability of whole architecture, there are two essential points that need to be focused.

(1) the results of the faults (e.g. system crash or SDC)

(2) the likelihood of the fault occurrence in the structure (e.g. size of the structure)

We classify the faults into three groups according to the way they manifest themselves: benign faults, catastrophic failures and errors (including SDCs). In result critical applications (e.g. financial applications), SDCs are the most critical faults for reliability in which the user must completely trust the result computed by the application (e.g. transferring the correct amount of money). On the other hand, some other applications are required to operate continuously (e.g. web servers). In these applications, catastrophic failures are more critical than SDCs. Thus, when FIMSIM calculates the vulnerability of the entire architecture, it multiplies the error rate and the catastrophic failure rate with C_e , C_c coefficients respectively. The weight of these coefficients are application dependent and defined by the user.

There are several aspects that affect the likelihood of the fault occurrence (LF) in a structure such as size, temperature, age, environmental conditions etc. In this study we consider two aspects. First, we give the higher weight to bigger structures according to the number of wires in the structures. Second, for the combinational logic (e.g. ALU) we multiply the likelihood with the probability that a fault injected to the inner state of the structure propagates to the output of the combinational logic without being masked inside the structure. We adopt these probabilities from a previous study of RTL analysis of the processor structures [13]. Note that, for more accurate LF calculation, other aspects can be considered as well.

We calculate the processor vulnerability (V) with the following formula:

$$V = \sum_i^{structures} \left(\left(C_e \times ER_i + C_c \times CR_i \right) \times LF_i \right)$$

$$LF_i = \ SR_i \times OR_i$$

$$ER_i = \frac{Number\ of\ Errors}{Number\ of\ Injected\ Faults}$$

$$CR_i = \frac{Number\ of\ Crashes}{Number\ of\ Injected\ Faults}$$

$$SR_i = \frac{Size\ of\ theStructure}{Size\ of\ the\ Entire\ Processor}$$

$$OR_i = The\, probability\, of\, fault\, propagation\, to\, the\, output$$

$$C_e = Error\ Coefficient$$

$$C_c = Crash\ Coefficient$$

Note that FIMSIM is flexible enough so that users can also generate their own processor dependability models.

Structure	SR	OR
Program Counter	0,003	1
Special Register File	0,4	1
Integer Register File	0,4	1
Instruction TLB	0,05	1
Data TLB	0,05	1
Arithmetic Logic Unit	0,08	0,7
ByPass Logic	0,003	1
	C_c	C_e
	1,0	1,0

TABLE I
NUMBER OF BITS IN MICROARCHITECTURE STRUCTURES AND THEIR AVERAGE TEMPERATURE.

IV. EVALUATION

In this section, we evaluate the vulnerability of in-order Alpha 21264 microarchitecture by utilizing our FIMSIM fault injection infrastructure and spec cpu2006 benchmark suite with test data set. 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 required for out-of-order execution such as the reorder buffer, issue queue and rename logic. We inject 100 faults per structure in each application to a random location in the structure (e.g a random bit of a randomly chosen register in the int-RF). For each fault model, we injected 14000 faults which is similar or very higher than prior fault injection analysis [12], [13], [14], [25].

Each of experimental results presented in this work represent the result of 20 applications from spec cpu2006 2000 fault injection per structure (20 application)

First, we generate checkpoints of the golden run at every 100M cycles after warming up 70M cycles. Then we inject faults at a random time (within 100M cycles) by loading the checkpoints by performing one injection per simulation. We injected faults to the first chunk since some applications in the spec cpu2006 benchmark suite does not have more than 200M instructions with test dataset. We calculate the vulnerability of the processor (V) by utilizing the formula that we explain in Section III-D. The size values of each structure and other related coefficients that we utilized in the formula, are presented in Table I.

Figure 3 presents the single fault injection results. Unsurprisingly, processors are vulnerable to permanent faults at most, as it is seen in the graph. 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

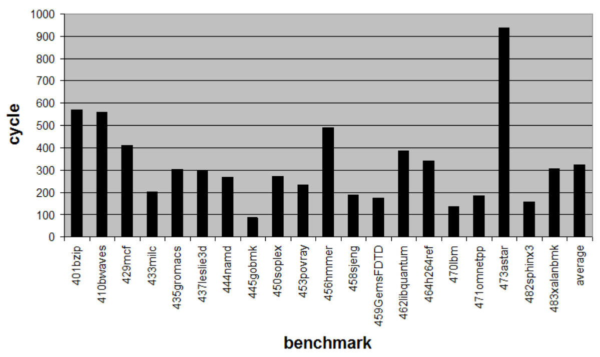

Fig. 5. Recovery time of FaulTM, a fault tolerance schema based on transactional memory

(transient or intermittent) are compensated. However, when there is a permanent fault in TLB, it is harmful for the whole architecture. Finally, the PC is the most vulnerable structure in these structure that any fault in the PC result in either error or system crash.

In Figure 4, we present the effects of multiple faults on the processor. Note that we can inject the vertical multi-bit faults only to the buffer structures (i.e. register files and TLBs). In the graph it is seen that 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. Bridging faults (dominant-0 and dominant-1) affecting the vulnerability in the similar way. Because the final result of the bit values changes if two bits are different from each other meaning they are effective in the same conditions.

Besides vulnerability reduction, recovery time of a fault tolerance schema is one of the essential criteria as well. In FIMSIM enables researchers to measure the recovery time as well. In Figure 5 we present the recovery time of FaulTM, a transactional memory based fault tolerance schema, as a case study. The recovery time of an error is based on the transaction size in FaulTM. For instance, if a fault affects a large transaction, the recovery time increases (i.e. astar, bzip, bwaves, hmmer).

V. RELATED WORK

Fault injection has been utilized to measure the dependability of the systems. Since simulation based fault injection provides the correlation between circuit level fault and its effects in hardware, it is very appealing for computer architecture designers.

Many simulation based fault injection simulates the models in Very high speed integrated circuits Hardware Description Language (VHDL). Gil et. al. [9] studied and compared these fault injection methods.

Nicholas J. Wang et. al. [27] implemented The Illinois Verilog Model (IVM) for injecting transient faults and characterizing the effects on a high performance processor pipeline.

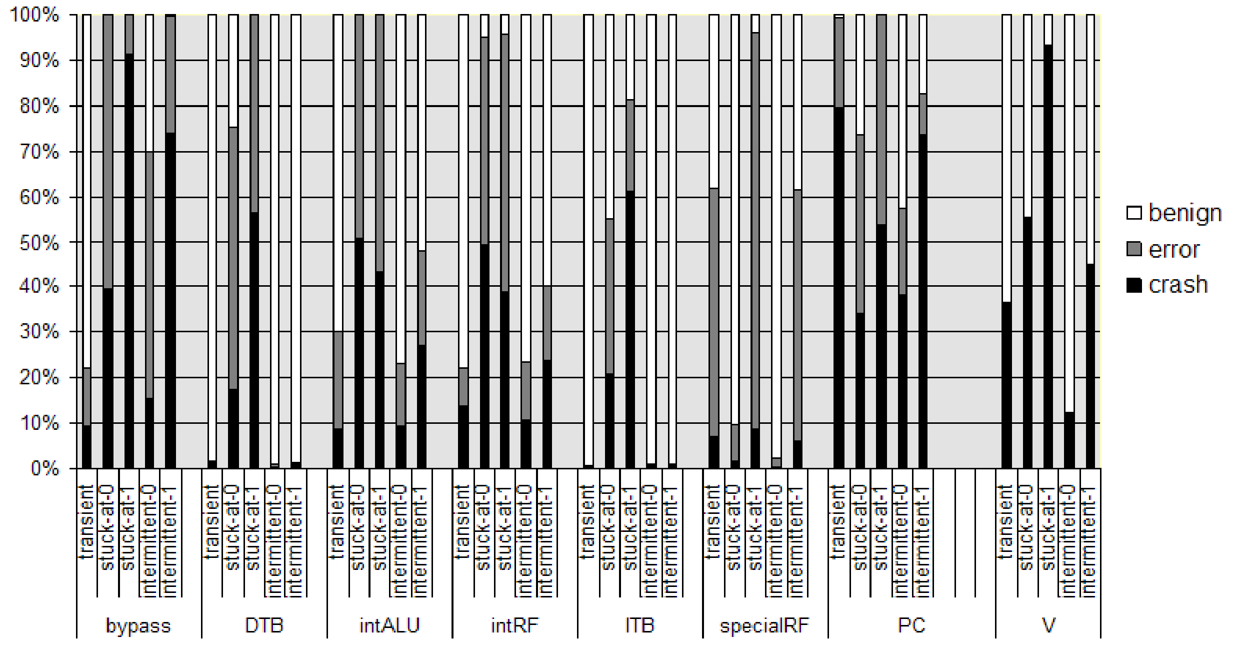

Fig. 3. Single Fault Injection to Spec2006 Benchmark.

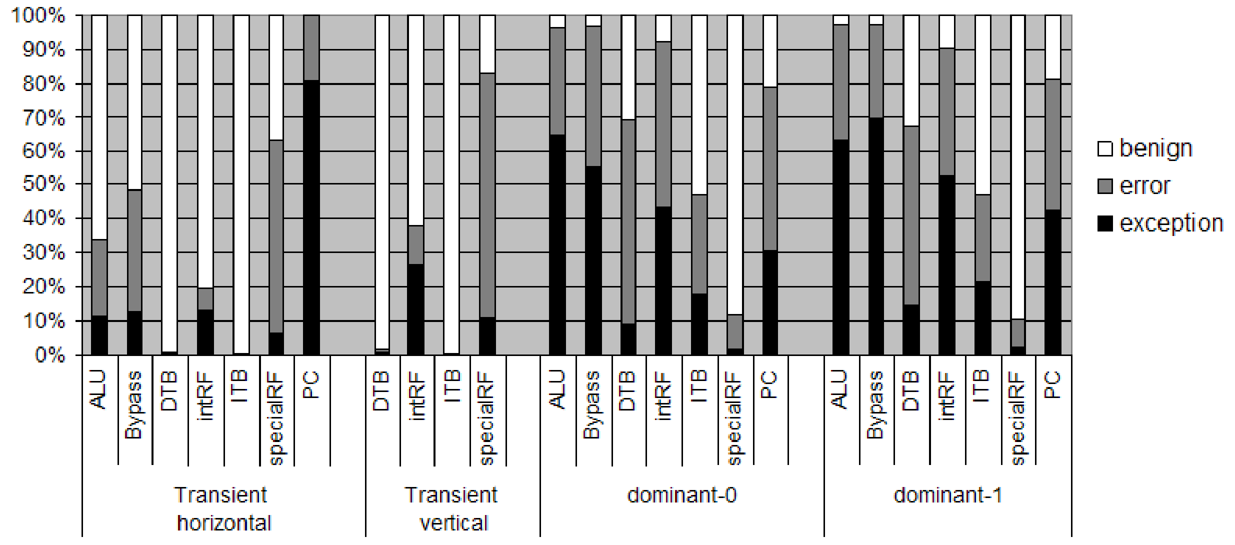

Fig. 4. Multi-bit Fault Injection to spec2006 benchmark.

They created a highly detailed RTL model of microprocessor. The results are traced by using uniform sampling.

Michail Maniatakos et. al. [15] pinpointed some limitations of IVM (e.g. can not implement the floating point instructions, inefficient to inject permanent or intermittent faults). They developed an extensive fault simulation infrastructure by interacting IVM with a functional simulator (simplescalar). Also, Man Lap Li et. al. [13] combined microarchitecture simulation with RTL level fault injectors to inject permanent faults to ALU, Decoder and Address Generation Unit (AGEN) to gain accurate fault injection results. However, these schemes are complex and slow for microarchitecture designers since they include RTL model.

Nishant J. George et. al. [8] injected single and double transient faults due to single particle strike. They injected faults to the register file and to the reorder buffer by using a functional simulator (PtlSim). The effects of the injected faults are classified according to the final results of the applications. Therefore, they could make their experiments only on short applications due to time constraints. Also, they can not distinguish whether a fault is benign or hidden in the architecture.

Man-Lap Li et. al. [14] injected permanent faults to a microarchitectur level full system simulation environment (GEMS+Virtutech Simics). They injected both single (stuck-at) and double (bridging) faults. Also, they pointed out the advantages of fault injection into microarchitecture level simulator (e.g. presenting a trade-off in speed and the ability to model long running workloads with OS activity).

So far, very few tries have been done in order to study the effects of intermittent faults by fault injection. Layali Rashid [19] injected intermittent faults in software level to understand their affects on the software. However, they made their experiments only on two applications (matrix multiply and insertion sort). Gracia et. al. injected intermittent faults in a VHDL based simulator [10]. To the best of our knowledge, there is no study about intermittent fault injection to the functional simulator.

In this study, we provide a complete fault injection tool on a functional full system simulator that can (1) inject transient, permanent and intermittent faults, (2) analyse the affects of the faults on the software since it is on full-system simulator, (3) inject single and multiple faults such as multiple bit flips due single or double particle strike, bridging or any combinations of different faults.

VI. Conclusion

In this study, we presented FIMSIM, a fault injection simulator in the microarchitecture level which takes the burden of designing and implementing of a fault injector from the researchers. FIMSIM is a compact tool that can inject transient, permanent or intermittent faults either as a single bit fault or as multi-bit faults. Therefore, it provides opportunity to comprehensively evaluate the vulnerability of microarchitectural structures. We also evaluate the vulnerability of in-order Alpha microarchitecture. We will publicly release FIMSIM at the same time with the presentation of the paper.

References

[1] *Alpha 21264 Microprocessor Hardware Reference Manual*. Compaq Computer Corparation, 1999.

[2] R. Baumann. Soft errors in advanced computer systems. *IEEE Design and Test*, 22:258–266, May 2005.

[3] N. Bidokhti. SEU Concept to Reality (Allocation, Prediction, Mitigation). In *Reliability and Maintainability Symposium (RAMS)*, 2010.

[4] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26:52–60, July 2006.

[5] C. Constantinescu. Impact of deep submicron technology on dependability of vlsi circuits. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 205–209, 2002.

[6] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23:14–19, July 2003.

[7] C. Constantinescu. Intermittent faults in vlsi circuits. In *Proceedings of the IEEE Workshop on Silicon Errors in Logic System Effects*, 2006.

[8] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach. Transient fault models and avf estimation revisited. *Dependable Systems and Networks*, 0:477–486, 2010.

[9] D. Gil, J. Gracia, J. C. Baraza, and P. J. Gil. 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.

[10] J. Gracia, L. J. Saiz, J. C. Baraza, D. Gil, and P. J. Gil. Analysis of the influence of intermittent faults in a microcontroller. In *Proceedings of the 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, 2008.

[11] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 34:1–17, 2006.

[12] N. Karimi, M. Maniatakos, A. Jas, and Y. Makris. On the Correlation Between Controller Faults and Instruction Level Errors in Modern Microprocessors. In *International Test Conference*, 2008.

[13] M. lap Li, P. Ramach, U. R. Karpuzcu, S. Kumar, S. Hari, and S. V. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults *. In *International Symposium of High-Performance Computer Architecture*, 2009.

[14] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 265–276, 2008.

[15] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris. Instruction-level impact analysis of low-level faults in a modern microprocessor controller. *IEEE Transactions on Computers*, 99, 2010.

[16] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt. Cache scrubbing in microprocessors: Myth or necessity. In *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 37–42, 2004.

[17] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *Proceedings of the International Symposium on Computer Architecture*, pages 99–110, 2002.

[18] R. Naseer and J. Draper. Parallel double error correcting code design to mitigate multi-bit upsets in srams. In *34th European Solid-State Circuits Conference*, pages 222–225, 2008.

[19] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. Towards understanding the effects of intermittent hardware faults on programs. *Dependable Systems and Networks Workshops*, 0:101–106, 2010.

[20] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. SWIFT: Software implemented fault tolerance. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 243–254, 2005.

[21] E. Rotenberg. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, page 84, 1999.

[22] J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe. Reunion: Complexity-effective multicore redundancy. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 223–234, 2006.

[23] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. Nowatzyk. Fingerprinting: Bounding soft-error detection latency and bandwidth. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 224–234, 2004.

[24] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of International Conference on Dependable Systems and Networks*, pages 177–186, 2004.

[25] N. J. Wang, A. Mahesri, and S. J. Patel. Examining ace analysis reliability estimates using fault-injection. In *34th Annual International Symposium on Computer Architecture*, pages 460–469, 2007.

[26] N. J. Wang, S. Member, and S. J. Patel. Restore: Symptom-based soft error detection in microprocessors. *IEEE Transactions on Dependable Secure Computing*, 3:188–201, 2006.

[27] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proceedings of International Conference on Dependable Systems and Networks*, pages 61–70, 2004.

[28] P. M. Wells, K. Chakraborty, and G. S. Sohi. Adapting to intermittent faults in multicore systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 255–264, 2008.

[29] G. Yalcin, O. Unsal, A. Cristal, I. Hur, and M. Valero. Faultm: Fault tolerance using hardware transactional memory. In *Workshop on Parallel Execution of Sequential Programs on Multi-core Architecture*, 2010.