Accelerated Microarchitectural Fault Injection-Based Reliability Assessment

Manolis Kaliorakis Sotiris Tselonis Athanasios Chatzidimitriou Dimitris Gizopoulos

Department of Informatics and Telecommunications
University of Athens, Greece
{manoliskal, tseloniss, achatz, dgizop}@di.uoa.gr

***Abstract*—Statistical Fault Injection on microarchitectural simulators can provide early and accurate reliability characterization for array based hardware components. Besides, microarchitectural fault injectors are easily configurable (facilitating many reliability studies) and orders of magnitude faster than RTL fault injectors, rendering them appropriate tools for early reliability estimation using large and realistic benchmarks. However, the throughput of the fault injection campaigns on microarchitectural simulators remains a bottleneck when a batch of campaigns must run for early reliability estimation of a processor (different microarchitectural characteristics, different workloads). This paper presents two different operation modes on top of a baseline framework for statistical fault injection campaigns, trading off between accuracy and speedup of the injection campaigns with a state-of-the-art out-of-order full-system x86-64 simulator as experimental vehicle. In the first mode, the injection experiments are stopped and classified as masked due to the following conditions: (i) the fault is over-written after the injection and it hasn't been read earlier, (ii) or the fault is injected on an invalid entry. The second mode has the same termination conditions as the first mode, but the injection experiments can also be terminated when an instruction that has read the faulty entry passes through the commit stage of the x86-64 out-of-order architecture. In the first mode, we observed a speedup up to 2.92x with no loss of accuracy in the vulnerability measurements for all structures. In the second mode an even higher speedup of up to 4.06x has been obtained with small loss in the accuracy of the vulnerability measurements.**

Keywords -early reliability evaluation; statistical fault injection; microarchitectural simulators; microprocessors

I. INTRODUCTION

Evolution in semiconductor devices leads to performance improvement using denser and more complex chips and more aggressive architectures with increased pipeline depth. But, the shrinking of transistors' size makes them prone to transient faults (soft errors) due to radiation and very low voltage operation [1] [2]. The need for early reliability estimation of the susceptibility of different hardware structures to this kind of faults is very important to avoid destructive malfunctions during the whole lifetime of the chip selecting the most appropriate protection mechanisms in terms of area and performance. For this reason, a great effort is spent to accurately predict the resilience of microprocessors to soft errors at the first stages of the design phase in order to avoid costly re-design cycles.

Microarchitectural full-system cycle accurate simulators [3] hold a dominant role in early and accurate reliability assessments of several array-based microarchitectural structures that occupy the majority of chip's area. This can be explained by the fact that they: (i) are available early enough in the design phase in contrast to the more detailed RTL and gate-level models, (ii) are faster than lower level models giving the ability to run plethora of large, complex and realistic benchmarks, (iii) consist of easily configured microarchitectural structures giving the ability to identify the most appropriate among them in terms of reliability, (iv) accurately model the effect of fault in array-based structures (in contrast to logic circuits and functional units), (v) allow the observation of the fault till the higher level of system stack and finally, (vi) are well maintained as they are extensively used in performance studies.

The existing reliability estimation approaches performed in microarchitectural simulators are summarized in three different domains: (i) probabilistic models [4] [5] [6], (ii) ACE (Architecturally Correct Execution) analysis [7] [8] [9] and (iii) fault injection [10] [11] [12] [13] [14] [15]. ACE analysis and probabilistic methods are faster than fault injection but lead to overestimations of vulnerability that can reach up to 7x and 3x, as presented in [12] and [16] respectively. On the other hand, fault injection approaches are slower but can accurately resemble the effect of a fault when it is focused on array-based hardware structures, providing accurate reliability characterization for these components. Another technique to speed fault injection up has been proposed in [18]; during a fault-free run it identifies faults that are overwritten and excludes them from the fault list.

Despite fault injection accuracy, execution time remains the major drawback of this technique considering that many campaigns must be completed early in design phase, for different components, microarchitectural characteristics, protection mechanisms and workloads. Except for leveraging parallelism of modern computing systems to run simultaneously multiple campaigns in multiple threads and workstations, statistical fault injection (SFI) [17] is also used to deliver quick and accurate estimations. Using lower confidence

978-1-4799-8606-4/15/$31.00

level and higher error margin, far less experiments could be executed in expense of accuracy. In this paper, we extend the baseline mode of an out-of-order cycle accurate full-system x86-64 fault injection framework with two extra modes of operation in order to speed up the fault injection campaigns. In the first mode, an injection experiment is forced to completion when the fault is overwritten before it is read and thus we classify it early and accurately as masked. In the second mode, an injection experiment is forced to completion when the fault is overwritten before it is read or when an x86 instruction reads the fault from the faulty entry and reaches the commit stage. The second method provides a tradeoff between speedup and accuracy in order to deliver a fast but not so accurate solution in the early reliability estimation problem.

II. METHODOLOGY

A. Fault Injection Framework

For the needs of our experiments we used an already introduced fault injection framework [10] based on MARSSx86, a state-of-the-art full-system cycle accurate x86-64 simulator [3]. The MARSSx86 simulator consists of the PTLsim CPU simulator [19] that includes all the details of the x86-64 out-of-order model and the QEMU emulator that provides all the characteristics of the full-system simulation. This framework is the most suitable for our studies as:

- It accurately simulates the x86-64 microprocessor along with its memory system [20].
- MARSSx86 and PTLsim have already been used for many reliability studies [10] [21] [22].
- It gives us the opportunity to trace the fault and its propagation from the hardware layer to the software layer of system stack.

The microprocessor's baseline configuration used in all the modes of framework's operation, resembles a modern out-of-order x86-64 microprocessor and is illustrated in Table I.

TABLE I. BASELINE CONFIGURATION

Component	Characteristics of Baseline Model
Pipeline	Out of Order
Physical Integer Register File	256 registers
L1 Data cache	32KB, write-back, 4-way set associative, 64B block size
L1 Instruction cache	32KB, write-back, 4-way set associative, 64B block size
L2 cache (unified)	1MB, write-back, 16-way set associative, 64B block size
LSQ (unified)	32 entries (16 load and 16 store entries)

In the context of our study, concerning the fault injection campaign speedup and the reliability assessment of several components of the x86-64 microprocessor, we used 7 benchmarks from the MiBench benchmarks suite [23]. These benchmarks are small enough to run them till the end, providing the probability to have an accurate characterization per structure and benchmark. Despite their small execution time, they have many similarities to SPEC benchmarks concerning the usage of major microarchitectural structures and have been previously used in many reliability studies [12] [24]. Table II presents the benchmarks used in this study and their committed instructions.

In this study, we carried out extensive fault injection campaigns of transient faults in the following structures of the microprocessor that hold the majority of chip's area:

- L1 Data cache
- L1 Instruction cache
- L2 unified cache
- Physical Integer Register File
- LSQ (data field)
- LSQ (address field)

TABLE II. BENCHMARKS AND COMMITTED INSTRUCTIONS

Benchmark	Committed x86 instructions
djpeg	6,372,580
search	145,000
smoothing	22,901,000
edges	2,296,000
corners	1,345,500
sha	12,450,000
qsort	16,097,000

Each statistical fault injection campaign consists of 2000 experiments and one golden run. A separate injection campaign was performed for each hardware structure, each benchmark and each of the three modes of operation. This number of experiments corresponds to 2.88% error margin and 99% confidence level according to [17]. A total number of 252,126 experiments (7 benchmarks x 6 structures x 2001 injections x 3 operation modes) were performed for the entire study.

B. Operation Modes and Classification

The proposed framework supports fault injection in three different operation modes: Full Execution (***Baseline***), Early Stop on Overwrite (***ESO***), Early Stop on Overwrite or first Read (***ESOR***).

All modes adopt different criteria and categories to assess fault injection's outcome[1]. Afterwards, we separately present the three operation modes with the used criteria and classes per mode.

[1] By the term outcome we mean the following: output of an application, exceptions that are generated during the execution of an application, standard error, time of execution etc.

1. Full Execution – Baseline

We run the application to the end and classify the outcome of a fault injection experiment in comparison with the outcome of a golden run. The classes used in this mode are the following:

Masked: The application's output and the raised exceptions of a fault injection experiment are equal to the ones of the golden run.

Silent Data Corruption – SDC: The application's output of a fault injection experiment differs from the one of the golden run but the raised exceptions are equal.

Detectable Unrecoverable Error – DUE: The raised exceptions of a fault injection experiment differ from the one of the golden run. We exclusively focus on the raised exceptions as an indicator of an error occurrence because the simulated model of the microprocessor doesn't provide any error detection or protection mechanisms.

Timeout: The application has not been completed within a reasonable period of time. We force the fault injection experiment to be completed and thus characterizing its outcome as timeout when the execution time of the application exceeds the 3x the execution time of the golden run.

***Crash*:** A fault causes an unrecoverable situation in which the program execution or the simulation process stops. A crash can be process crash (abnormal termination of the program), system crash (kernel panic), or simulator crash.

Assert: The case in which the simulator reaches an abnormal state due to the fault, raising an assertion to stop the simulation as it is unable to handle the situation.

In essence, a fault injection experiment runs to completion only in case that belongs to *masked*, *SDC* or *DUE* category. Moreover, the sum of *SDC*, *DUE*, *Timeout*, *Crash* and *Assert* categories corresponds to the vulnerability of the structure.

2. Early Stop on Overwrite – ESO

We run the application to the end except for the cases that the fault is overwritten before it is read or the fault is injected on an invalid entry. The former mode (*Baseline*) has been extended to identify if the fault is masked prior to its use. In such cases, we can safely characterize the fault injection experiment as masked and thus stop it early, before the simulation's completion. The extra logic raises safely an assertion message handled by the parser described in [10].

Thus, the former classification (*Baseline*) was extended with the following sub-classes for the masked category: ***Write After Injection – WAI*** (overwritten or injected on an invalid entry) and ***not Write After Injection – not WAI*** (masked but not-overwritten or not injected on an invalid entry).

3. Early Stop on Overwrite or first Read – ESOR

We run the application to the end except for the cases that the fault is overwritten or is read by a committed instruction. The former mode (*ESO*) has been extended to identify when corrupted data (in presence of fault) are read. The extra logic raises an assertion when an instruction reads the faulty bit and passes through the commit pipeline stage, meaning that the architectural state has been corrupted by the fault. This mode of operation consists of the next classes:

***Read After Injection - RAI*:** The faulty bit is read in the execution stage by an instruction and then this instruction reaches commit stage. Undoubtedly, the early stop of experiments right after the use of corrupted data limits our potential for accurate reliability characterization because we ignore any masking on the software level.

***Write After Injection - WAI*:** The faulty bit is written or the fault is injected on an invalid entry. Thus, the fault experiment is safely classified as masked in both cases.

***Unknown*:** The corrupted data (due to the fault) are moved from the target structure to another structure and are still potentially harmful for the system. These cases appear in L1 data cache and in the unified L2 cache of our studied structures as they both have write-back policy. For instance, a cache line of a lower cache level or memory updates its data with a corrupted block of data that was evicted by a higher level cache. This block could influence the correct execution of the program during the rest of its execution, but we don't trace the fault propagation through the memory system.

***Unused*:** The outcome of the experiment cannot be classified to any of the above categories i.e. the moment of injection is close to the completion of the experiment and its impact never manifests or the fault is injected on an unused by the program entry.

Moreover, the sum of *WAI* and *Unused* categories defines a conservative lower boundary of the overall masking probability of a structure, while the sum of the *RAI* and *Unknown* categories represents the overestimated vulnerability of the structure.

We enhanced the potential of our framework to support *ESO* and *ESOR* modes in order to speed up fault injection campaigns. Especially, fault injection on *ESOR* mode is a tradeoff between speedup and accurate reliability estimation because the overestimation of vulnerability is inevitable. Table III summarizes the vulnerability functions used in all modes of operation.

TABLE III. VULNERABILITY ACROSS MODES

Operation Mode	Vulnerability Function
Baseline	1 – *Masked*
ESO	1 – *Masked*
ESOR	1 – *WAI* – *Unused*

Moreover, if an experiment involves a fault injection close to the completion of the application, then probably we will have negligible or no speedup at all. Fig. 1 summarizes how we can speed up a fault injection experiment in the *ESO* and *ESOR* modes of operation.

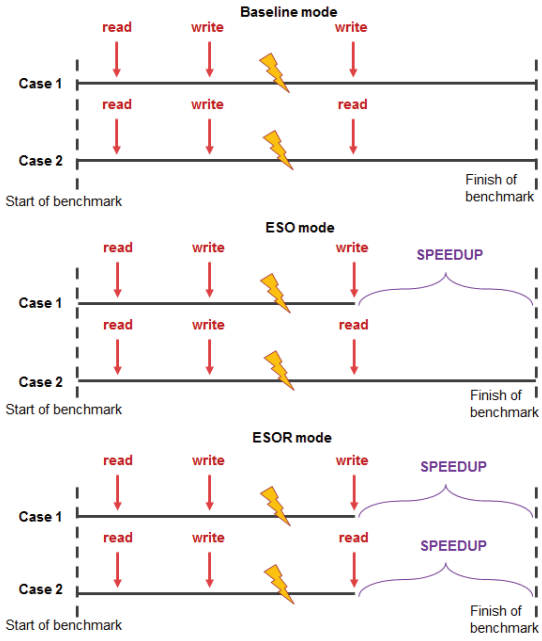

Figure 1. Speedup in the ESO and ESOR modes of framework operation.

The categories of *Baseline* and *ESO* mode are equivalent but this is not the case for the categories of *ESOR* mode. The *WAI* and *Unused* categories of the *ESOR* mode certainly lead to *Masked* but the *Unknown* and *RAI* categories may result in any category of the *Baseline* or *ESO* modes. Fig. 2 illustrates the correlation of categories among the three modes.

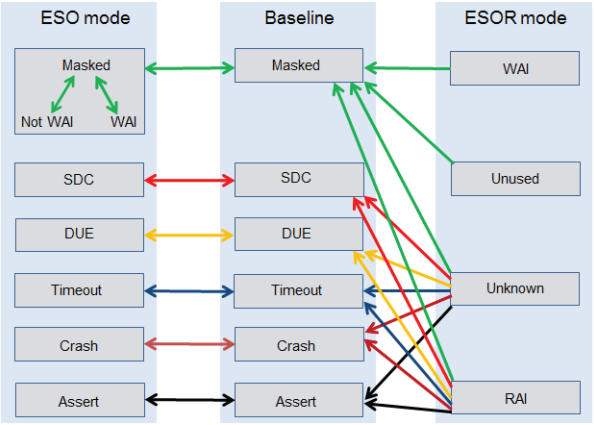

Figure 2. Correlation of classes among the three modes of operation.

III. Experimental Results

In this part, we present and analyze the results of the aforementioned three modes of operation (*Baseline* full-execution mode, *ESO* and *ESOR*), in terms of accuracy of the final reliability estimation and speedup compared to the *Baseline* full-execution mode. Each column in the following graphs corresponds to one microarchitectural structure, representing the average values obtained from executing statistical fault injection campaigns running the 7 benchmarks illustrated in Table II.

A. *Full Execution – **Baseline** mode*

The first part of our analysis concerns the *Baseline* mode, where the fault injection experiments run to the end. The results for the six structures of our study are presented in Fig. 3. It is observed that the first level caches are the most vulnerable among the structures, while the unified L2 cache is the most reliable. The vulnerability of L1D, L1I and L2 cache is 14.95%, 9.95%, 2.08% respectively. Despite the fact that data in L2 cache may present more residency than in first level caches, the result of more reliable L2 cache can be explained because cache blocks are more often used from first level caches than L2 cache, increasing the probability of fault propagation to the core. Furthermore, the evicted blocks from the write-back L1D cache can probably over-write the faults that were injected on L2.

Moreover, in L1D cache *SDC* category dominates as it hosts data that can corrupt silently the output of the program, while L1I cache holds mostly data that their corruption can probably lead to crashes or assertions and termination of the experiment. In Integer Physical Register File and L2 cache the not-masked categories are well-balanced, while in LSQ the *Assert* class dominates among the not-masked categories. The vulnerability of Register File, LSQ (data field) and LSQ (address field) is 2.86%, 2.60% and 3.78% respectively.

The prevailing category is the *Masked* across all structures and ranges from 85.05% to 97.92%. In general, *Masked* category consists of faults that are overwritten at microarchitecture level or application level. Especially, the speedup of the proposed modes stems mainly from microarchitectural level masking.

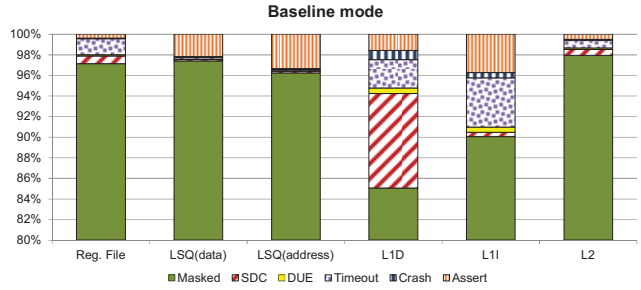

Figure 3. Faulty Behaviors classification of *Baseline* mode.

B. *Early Stop on Overwrite – **ESO** mode*

In this mode of operation the experiment is stopped only when the fault is over-written before it is read or the injection is targeted on an invalid entry. Consequently, *ESO* mode decreases the simulation execution time without sacrificing the accuracy of reliability estimation. For this reason, the

reliability classification of this mode is exactly the same with that of the *Baseline* mode, illustrated in Fig. 3.

In essence, the degree of speedup in our fault injection campaigns using *ESO* mode is strongly related to the percentage of experiments that can be definitely characterized as masked, allowing the termination of the experiment before its completion, without any loss of accuracy. In Fig. 4, the *WAI* class represents the experiments that can be safely stopped before the end of the experiment and classified as masked. The worst case is that of L2 cache that has only 11.71% *WAI* experiments, while Integer Physical Register File and LSQ (address field) present the highest amount of *WAI* (91.36% and 93.95% respectively). Moreover, the *WAI* category in LSQ (data field), L1D and L1I is 43.20%, 46.71% and 62.84% respectively. The low percentage of *WAI* category observed in the caches is related to the access patterns of the workload. In our experiments, L1D and L1I have more over-written faults as they both have more blocks coming from the lower level of memory hierarchy and L1D has more store hits than L2 cache. The Register File has a high percentage of over-written faults, because many injections took place in a period of time when the data of the register weren't useful. Finally, the *WAI* class of LSQ (data field) is higher than that of LSQ (address field), due to the forwarding of data from store entries to load entries when a dependency exists between them.

The major conclusion coming up from Fig. 4 is that for all the structures except for L2 cache there is great potential of injection campaign speedup using *ESO* mode of operation. This is based on the fact that the more the over-written faults, the more speedup will be gained during the fault injection campaign.

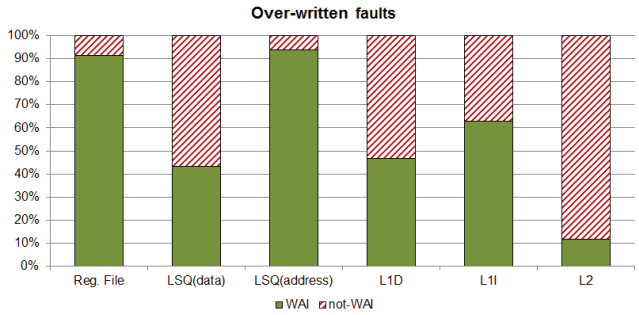

Figure 4. Percentage of over-written or injected on invalid entry faults.

C. *Early Stop on Overwrite or first Read – **ESOR** mode*

The third mode of operation balances between reliability estimation accuracy and speedup of the injection campaign. Thus, this mode of operation must be evaluated in terms of accuracy and speedup compared to the full execution of the entire campaign (*Baseline* mode). Fig. 5 presents the reliability classification of the *ESOR* mode for all structures and benchmarks. Note that the reliability of a structure in this mode consists of the percentage of *WAI* and *Unused* category, while the *RAI* and *Unknown* categories can be potential not-masked, corrupting the correct execution of the program. Another important note concerning speedup is that *Unused* category ensures the accuracy of the estimation, but it does not contribute to speedup, as the experiment runs to the end for this case.

Moreover in Fig. 5, it is observed that the three caches consist of a high percentage of *Unused* category (~24% for the first level caches and 79.29% for the L2), omening that the speedup of caches will be limited by this factor. Conversely, the high percentage of *WAI* omens a good speedup for the cases of Integer Physical Register File and LSQ, which present a percentage of more than 86% in their *WAI* category. The percentage of *RAI* for all structures is less than 10%, except for L1D that has 11.83% and L1I cache with 23.69%. Finally, L1D cache presents a high percentage (~23%) of the potentially not-masked category (*Unknown*), as there are many evictions of dirty blocks from the L1D cache to the lower memory levels, but this is less intense in L2 cache with only 9.85% *Unknown*.

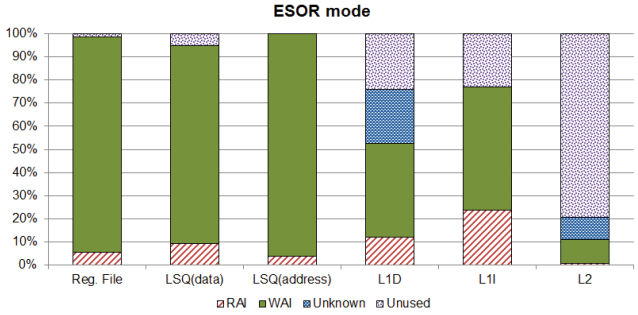

Figure 5. Faulty Behaviors classification of *ESOR* mode.

Based on the definition of vulnerability for the three modes of operation as illustrated in Table III we evaluate the inaccuracy of the *ESOR* mode of operation as presented in Fig. 6. The inaccuracy between *Baseline* and *ESOR* mode is only 2.66% in the Integer Physical Register File, 0.10% in the address field of LSQ, 6.58% in the data field of LSQ and 8.47% in L2 cache. On the contrary, the inaccuracy in the L1D is 20.13% and in the L1I is 13.74%.

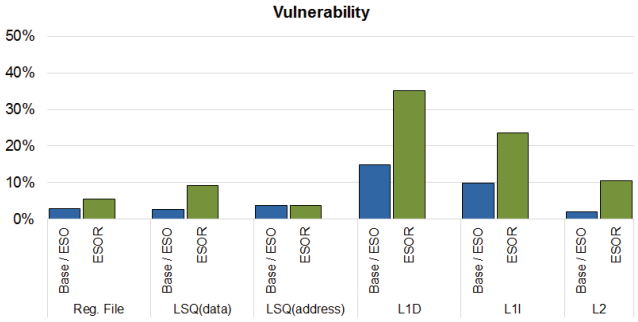

Figure 6. Structures vulnerability reported by the three operation modes. There is no loss of accuracy in the vulnerability reports between the baseline mode and the ESO mode, while ESOR mode reports higher vulnerability in all cases.

Fig. 7 summarizes the speedup of the three operation modes. We can observe that speedup scales from *Baseline* mode to *ESO* mode and from *ESO* mode to *ESOR* mode in all cases. The address field of LSQ presents the best scaling and the best speedup among all structures (2.92x in *ESO* mode and 4.06x in *ESOR* mode) and this is justified by the highest WAI rate that it features according to Fig. 4 and Fig. 5. The worst scaling is presented in L2 cache (~1.06% for both *ESO* and *ESOR* modes), but this can be explained by the high percentage of *Unused* category illustrated in Fig. 5. In general, all the caches that present a high percentage of the *Unused* category do not speedup so well as the structures with low percentage of *Unused* category. Combining the results presented in Fig. 6 and Fig. 7, we conclude that for the intra core structures (Physical Integer Register File, address and data fields of LSQ), the best solution to speed up the statistical fault injection campaign is the third mode of framework's operation (*ESOR* mode) with negligible loss of estimation accuracy, getting a high speedup of 3.38x, 4.06x and 3.37x respectively. Except for *ESOR* mode, the *ESO* mode could be also used without any accuracy loss getting speedup of 2.63x, 2.92x and 1.46x respectively.

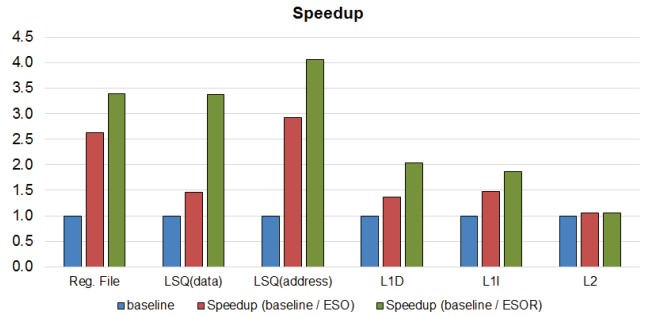

Figure 7. Speedup of the three operation modes.

On the other hand, the best choice for an architect to estimate the reliability of caches is the *ESO* mode of framework's operation. This conclusion comes from the fact that the inaccuracy of caches' reliability assessment using *ESOR* mode is not negligible (from 8.47% for L2 cache to 20.13% for L1D cache) and the speedup is not as high as in the intra core structures (from 1.06% for L2 to 2.04% for L1D cache). Consequently, *ESO* mode is the best choice for caches to speedup campaign (with 1.37x, 1.48x and 1.05x speedup for the L1D, L1I and L2 respectively) and ensure the estimation accuracy.

IV. Conclusions

In this study, we proposed two modes of operation built on top of an x86-64 out-of-order cycle accurate full system simulator, in order to speed up the early reliability assessment problem of modern high performance processors. Both methods were evaluated in terms of accuracy and speedup compared to the *Baseline* full execution of a statistical fault injection campaign. We conclude that for the hardware structures inside the CPU (Integer Physical Register File and the LSQ) the speedup achieved is up to 4.06x with negligible loss of accuracy, while for the caches the speedup is up to 2.04x which comes with a higher but acceptable loss of accuracy in the reliability estimation.

Acknowledgment

This work is supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404, and also by EU and Greek national funds under the Thales/HOLISTIC project and the DIaSTEMA project.

References

[1] C.Constantinescu, "Trends and challenges in VLSI circuit reliability", IEEE Micro, vol. 23, pp. 14-19, July 2003.
[2] S. Nassif, N. Mehta, Y. Cao, "A resilience roadmap", DATE 2010.
[3] A. Patel, F. Afram, S. Chen, K. Ghose, "MARSS: A full system simulator for multicore x86 CPUs", DAC 2011.
[4] A. Savino et al., , "Statistical reliability estimation of microprocessor-based systems", IEEE Transaction on Computers, vol. 61, no. 11, pp. 1521-1534, September 2012.
[5] J. Suh, M. Annavaram, M. Dubois, "MACAU: A Markov model for reliability evaluations of caches under single-bit and multi-bit upsets", HPCA 2012.
[6] X. Li, S. V. Adve, P. Bose, J. A. Rivers, "SoftArch: An architecture-level tool for modeling and analyzing soft-errors", DSN 2005.
[7] A. Biswas et al., "Computing architectural vulnerability factors for address-based structures", ISCA 2005.
[8] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor", MICRO 2003.
[9] A. A. Nair, S. Eyerman, L. Eeckhout, L. K. John, "A first-order mechanistic model for architectural vulnerability factor", ISCA 2012.
[10] N. Foutris, M. Kaliorakis, S. Tselonis, D. Gizopoulos, "Versatile architecture-level fault injection framework for reliability evaluation", IOLTS 2014.
[11] H. Cho, S. Mirkhani, C.-Y. Cher, J. Abraham, S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design", DAC 2013.
[12] N. George, C. Elks, B. Johnson, J. Lach, "Transient fault models and AVF estimation revisited", DSN 2010.
[13] S. Feng, S. Gupta. A. Ansari, S. Mahlke, "Shoestring: propabilistic soft error reliability on the cheap", ASPLOS 2010.
[14] S. K. S. Hari, R. Venkatagiri, S. V. Adve, H. Naemi, "GangES: Gang error simulation for hardware resilience evaluation", ISCA 2014.
[15] S. K. S. Hari, S. V. Adve, H. Naemi, P. Ramachandran, "Relyzer: Exploting application-level fault equivalence to analyze application resiliency to transient faults", ASPLOS 2012.
[16] N. J. Wang, A. Mahesri, S. J. Patel , "Examining ACE analysis reliability estimates using fault injection", ISCA 2007.
[17] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence", DATE 2009.
[18] A. Miele, "A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems", Microprocessor and Microsystems – Embedded Hardware Design, vol. 38, no. 6, pp. 567-580, June 2014
[19] M-T. Yourst, "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator", ISPASS 2007.
[20] J. Stevens et al., "An integrated simulation infrastructure for the entire memory hierarchy: cache, DRAM, non-volatile memory, and disk", Intel Technology Journal, vol. 17, no. 1, 2013.
[21] M-T. Chang, P. Rosenfeld, S-L. Lu, B. Jacob, "Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM", HPCA 2013.
[22] A. Mayberry, M. Laquidara, C. Weeds, "Characterizing the microarchitectiral side effects of operating system calls", ISPASS 2013.
[23] M. R. Guthaus et al., "MiBench: A free commercially representative embedded benchmark suite", IWWC 2001.
[24] A. A. Nair, L. K. John, L. Eeckhout, "AVF Stressmark: Towards an automated methodology for bounding the worst-case vulnerability to soft errors", MICRO 2010.