Microprocessor Reliability-Performance Tradeoffs Assessment at the Microarchitecture Level

Sotiris Tselonis Manolis Kaliorakis Nikos Foutris George Papadimitriou Dimitris Gizopoulos
Department of Informatics & Telecommunications, University of Athens, Greece
{tseloniss, manoliskal, nfoutris, georgepap, dgizop}@di.uoa.gr

***Abstract*—Early decisions in microprocessor design require a careful consideration of the corresponding performance and reliability implications of transient faults. The size and organization of important on-chip hardware components such as caches, register files and buffers have a direct impact on both the microprocessor resilience to soft errors and the execution time of the applications. In this paper, we employ a state-of-the-art x86-64 full-system micro-architectural simulator and a comprehensive fault injection framework built on top of it to deliver a detailed evaluation of the reliability and performance tradeoffs for major hardware components across several important parameters of their design (size, associativity, write policy, etc.). We also propose a simple and flexible fitness function that measures the aggregate effect of such design changes on the reliability and the performance of the studied workload.**

Keywords— microprocessors, reliability estimation, transient faults, performance, fault injection

I. INTRODUCTION

Despite the continuous performance improvement in modern microprocessor chips, a known downside of technology scaling is the increased susceptibility of hardware structures to soft errors [1] [2] [3]. Design decisions related to the parameters of such key hardware components (size or associativity of a cache memory or size of a physical register file) should be both performance and reliability aware, because both aspects are workload and microarchitecture dependent [4].

Both microarchitectural and RTL simulators are available at the early stages of a microprocessor design. RTL simulations are more accurate but have very low throughput. On the other hand, cycle accurate microarchitectural simulators [5] offer the potential of making performance and reliability evaluations at higher levels of abstraction but at a significantly higher throughput, helping engineers to make design decisions early enough at the design stage. While microarchitectural simulators provide a set of measurements for performance, they do not provide relevant support for reliability studies. A major missing part of the microprocessor models in simulators is that of the data/instruction arrays in caches, improving simulation time but rendering reliability studies through fault injection impossible. Thus, extensions in these structures are inevitable to make fault injection experiments feasible [6] [7].

Major storage arrays of a microprocessor (caches, register files, queues, buffers etc.) are of prevalent importance since occupy very large parts of the chip's silicon estate and largely determine the overall microprocessor reliability. Several microprocessor reliability studies are used for early reliability assessments based on ACE (Architecturally Correct Execution) analysis [8] [9] [10] [14], probabilistic models [11] [12] or fault injection [6] [7] [13] [14] [15]. Probabilistic models as well as ACE analysis are undoubtedly faster than fault injection, but they overestimate the vulnerability of microprocessor structures by several times [14] [16]. On the other hand, fault injection on top of microarchitectural simulators resembles much closer the actual behavior of hardware and software in the presence of faults [6] [14]. Statistically significant fault injection campaigns on such infrastructures deliver very accurate reports on the faulty behavior of hardware components [17] at the expense of larger simulation times.

In this paper we present a complete fault injection analysis of transient faults which jointly considers the reliability and the performance impact of several important design parameters on a modern out-of-order x86-64 architecture. We focus on the most important storage arrays of the microprocessor: the physical register file, the cache memories (split L1 data and L1 instruction cache and the unified L2 cache) and the load/store queue. Starting from a baseline configuration for all the components of our study, we modify the sizes of the hardware structures and assess the impact on reliability and performance. For each cache memory structure we also study the impact of the different write policies (write back vs. write though) while for the important case of the L1 caches (both instruction and data), we extensively evaluate the impact of different associativity points as well as their behavior in the presence or absence of L1 prefetchers. To our knowledge, no study of this extent that considers all these parameters has been reported previously.

Each different design point is analyzed based on an individual statistically safe fault injection campaign. Along with the performance information for each configuration a microprocessor designer can make informed decisions about the hardware protection mechanisms required for a particular configuration and workloads [10]. To assist design decisions we define a simple yet flexible fitness function which describes the combined effect on reliability and performance that certain design parameters have.

II. EXPERIMENTAL SETUP

A. The Fault Injection Framework

In our study, we used MaFIN a recently introduced and easily configured fault injection framework [6] [7] (with implemented data arrays on caches and complete fault

978-1-4673-8454-4/16/$31.00 ©2016 IEEE

injection infrastructure) that is built on top of MARSSx86 microarchitectural full-system x86-64 simulator [5]. We further extended the memory hierarchy model with a prefetch-on-miss sequential prefetcher in the L1 instruction cache and a stride prefetcher in the L1 data cache [18] to better resemble a modern x86-64 processor. MARSSx86 and its predecessor (PTLsim) have been extensively used in many performance and reliability studies [7] [15] [19].

We selected MARSSx86 because: (i) it accurately simulates an x86-64 out-of-order microprocessor model along with its memory system [20], (ii) it is full-system and cycle-accurate, (iii) it consists of easily configurable microarchitectural structures, giving us the opportunity to combine both performance and reliability studies in the same tool framework.

B. Hardware Structures Configuration

We use fault injection and modify a single parameter of the baseline microarchitecture (presented in TABLE I) at a time, to monitor the impact of individual changes on the reliability of the structure in conjunction with the performance of the application. The variants of the baseline model used in our experiments are illustrated in TABLE II. In the first two columns we refer to the target structures and their microarchitectural features while in the last column we list the alternative values that each feature had in our experiments.

TABLE I BASELINE CONFIGURATION

Structure		Baseline Model
Pipeline		OoO
Issue Queue		32 entries
Reorder Buffer		64 entries
Phys. Int. Reg. File		256 registers
L1 D cache		32KB, 64B, 4-way, write-back, stride pref.
L1 I cache		32KB, 64B, 4-way, write-back, sequential pref.
Unified L2 cache		1MB, 64B, 16-way, write-back, w/o pref.
Unified LSQ		32 entries (16 load queue, 16 store queue)
Branch Predictor		Tournament Predictor
Branch Target Buffer	Direct branches	1K entries, 4-way
	Indirect branches	512 entries, 4-way
RAS		16 entries
Functional Units		2 Integer ALUs, 2 FP ALUs, 4 AGU

C. Metrics

We apply a fault mask during a fault injection run and classify its outcome in one of the following categories, comparing the application's output, the number of raised x86-64 exceptions and the execution time with the golden ones.

Masked: The application's output and the raised exceptions are equal to the golden ones.

Silent Data Corruption (SDC): The application's output differs from the golden one but the raised exceptions are equal to the golden ones.

Detected Unrecoverable Error (DUE): The number of raised exceptions is greater than the golden one. We focus on the raised exceptions as an indicator of an error occurrence due to the lack of any error detection or protection mechanism in the simulated model of microprocessor.

Timeout: Program flow has been trapped and the simulator stopped committing instructions (deadlock) or has been redirected on random code areas executing instructions (livelock). This timeout limit was set in three times the fault-free execution time.

Crash: The fault causes process, system (such as kernel panic) or simulator crashes.

Assert: Simulator termination due to assert function.

TABLE II EXPERIMENTAL SETUP

Structure	Parameter	Values
Phys. Register File	Number of Registers	64/128/256
L1 D cache	Size	16KB/32KB/64KB
	Associativity	1/2/4/8
	Write policy	WB/WT
	Prefetcher	enabled/disabled
L1 I cache	Size	16KB/32KB/64KB
	Associativity	1/2/4/8
	Write policy	WB/WT
	Prefetcher	enabled/disabled
L2 cache	Write policy	WB/WT
LSQ	Number of entries	32/64/96

For the performance and the reliability evaluation of the benchmarks we use the IPC (instructions per cycle) metric and the FIT (failures in time) rates respectively. In all our experiments, we assume an arbitrary raw FIT rate of 0.01 per bit, but different rates can also be used. We also define the *fitness* metric that quantifies the impact of microarchitectural changes on both reliability and performance:

$$fitness = a \times \frac{1}{FIT} + (1 - a) \times IPC \qquad (1)$$

In equation (1), FIT is the fraction of the failures in time (on average for all benchmarks) that a hardware component with a specific configuration has over the one of the baseline model ($FIT=FIT_{conf}/FIT_{base}$); therefore, FIT >1 means that the studied configuration has a higher FIT rate (smaller reliability) than the baseline configuration. Similarly, IPC is the fault free committed instructions per cycle (on average for all benchmarks) with a specific hardware configuration over the one with the baseline hardware configuration ($IPC=IPC_{conf}/IPC_{base}$); therefore, IPC>1 means that the studied configuration is faster than the baseline. Parameter *a* is designer-defined (taking values from 0 to 1) and represents a wide range of designs that put more emphasis on the reliability or on the performance or balances both. The smaller the value of *a*, the more importance is given to performance (IPC). On the contrary, the larger the value of *a*, the more importance is given to reliability. If *a* equals to 0.5, then the same importance is given to both performance and reliability. Consequently, every fitness value of equation (1) represents a design point corresponding to an experimental setup that can be either better (in terms of reliability and performance) than the baseline configuration ($fitness_{conf} > fitness_{base}$) or worse ($fitness_{conf} < fitness_{base}$), where $fitness_{base} = 1.00$.

III. EXPERIMENTAL RESULTS

In our experiments, we focused on individual deviations from the baseline configuration, modifying only one configuration parameter at a time to assess its effect on reliability and performance.

A. Fault Sampling

For each injection campaign we used the formula of [17] which provides the number of injection experiments required

given the following inputs: (i) the size of a hardware structure in bits, (ii) the execution time of a benchmark in cycles, and (iii) the required statistical confidence level and error margin. Among these inputs confidence and error margin mainly affect the number of required fault injections. We ran 2000 fault injection experiments for each campaign corresponding to 2.88% error margin and 99% confidence level.

B. Benchmarks

Performance and reliability evaluations are workload dependent [4] [10] and a careful selection of benchmarks is vital for the accuracy of a study. We carried out fault injection campaigns during the execution of 7 benchmarks (*search, corner, edge, sha, qsort, smooth*) from MiBench suite because their short execution time enables us to execute them completely and observe the effect of a fault in the program's output [21]. Moreover, they are representative of real world applications (having many similarities to SPEC benchmarks i.e. instruction mix, throughput) and they also have been previously used in many reliability studies [7] [14] [15] [22].

C. Characterization of results

Our extensive fault injection campaigns of single transient faults target important microarchitectural structures (L1 Data cache, L1 Instruction cache, L2 unified cache, physical integer register file, and LSQ) across different values of their parameters shown on TABLE II. In total 336,000 fault injections have been executed (7 benchmarks x 24 different parameters combinations x 2000 injections per campaign).

We present our findings in two subsections. The first one presents the reliability characterization of the hardware structures at the different design points. In the second, we apply the fitness metric to correlate the performance and the reliability for all the variations of the components from the baseline model.

a) Reliability Characterization

In the bars from Figure 1 to Figure 11, we present the fault effect classification on average and per benchmark for each parameter of the hardware components (size in all components, associativity of L1 caches, write policy of all caches and behavior of L1 caches with or without prefetcher). Moreover, the second row of TABLE III to TABLE VII presents the Failures In Time (FIT) for each aforementioned parameter of the hardware components. In essence, a structure's reliability is inversely proportional to FIT and depends on the raw fit rate, the number of structure's bit and the vulnerability factor (percentage of not masked categories). The most vulnerable component is the L2 cache with 2318.9 FIT for the baseline model (TABLE V) and the most reliable is the LSQ with 0.5 FIT for the baseline model (TABLE VII).

Hardware Structures Size

Firstly, we focus on the sizes of the L1 caches, the physical register file and the LSQ. Figure 1 to Figure 4 show the faulty behavior classification for these four structures per benchmark and on average for the 7 benchmarks.

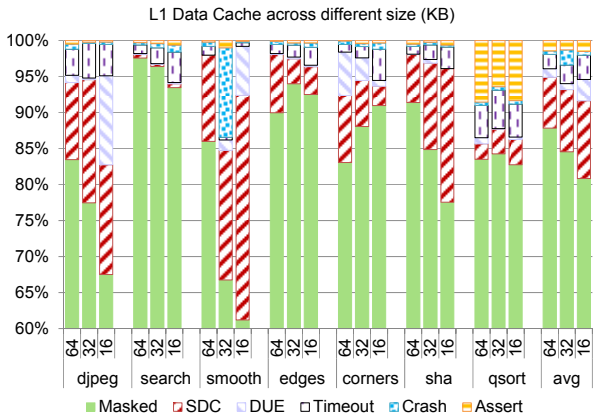

Figure 1: Faults classification in L1 Data cache (sizes).

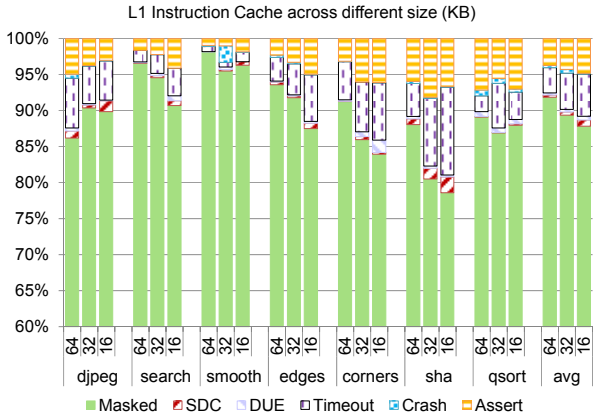

Figure 2: Faults classification in L1 Instr. cache (sizes).

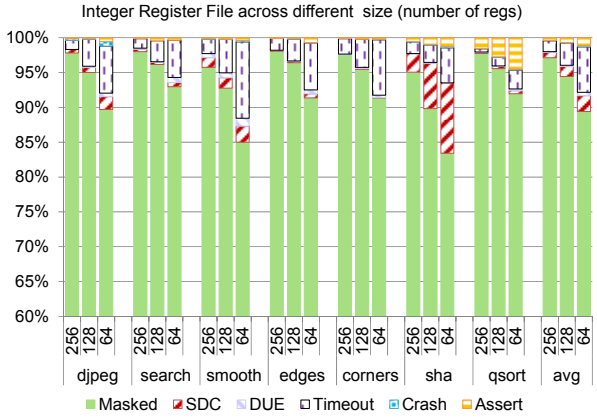

Figure 3: Faults classification in Physical Register File (sizes).

In the two L1 caches (Figure 1 and Figure 2), there are some benchmarks with opposite behavior but the average trend shows an increase of percentage of masked class for larger sizes: the average masked class of L1 Data cache increases by 7 percentage points and the one of L1 Instruction cache increases by 5 percentage points from 16KB to 64KB. In the register file (Figure 3), all benchmarks follow the same trend featuring higher percentage of masked class when the register file contains more registers and the average percentage of masked class of register file increases by 7 percentage points from 64 to 256 registers. The LSQ (Figure

4) features a smaller but still important 2 percentage points increase in the percentage of masked category from 32 to 96 entries.

All structures follow the trend to feature less FIT rates (more reliable) for smaller sizes because the size of a structure is more dominant than the vulnerability factor (percentage of not masked categories) in the computation of FIT. Especially, the highest reliability (the least FIT rate) is observed for 16KB L1 Data cache (251.1 FIT in TABLE III), 16KB L1 Instruction cache (159.4 FIT in TABLE IV), physical register file of 64 registers (4.3 FIT in TABLE VI) and LSQ of 32 entries (0.5 FIT in TABLE VII). The trend of both first level caches is similar to the reported findings in [10], which are based on ACE analysis and use SPEC2000 benchmarks.

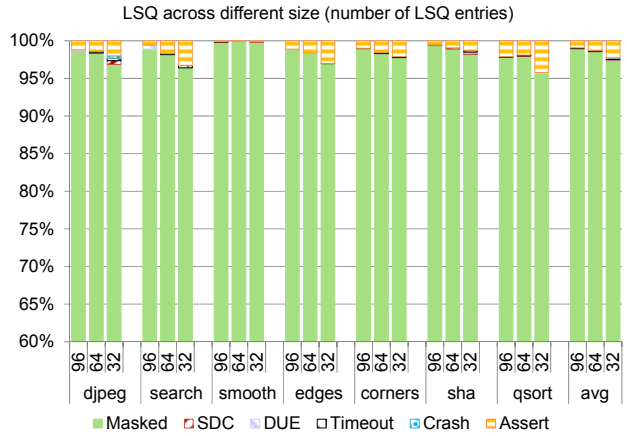

Figure 4: Faults classification in LSQ (sizes).

Caches Associativity

In this part, we study the L1 caches for different associativity values. In Figure 5, the average percentage of masked category of L1 Data cache is almost insensitive to associativity while the percentage of masked category of only two benchmarks (djpeg and smooth) features changes for different associativity. In Figure 6, the masked category of L1 Instruction cache increases by 6 percentage points from a direct-mapped to an 8-way set associative cache (for the same size of 32KB).

In general, the most reliable is the 2-way set associative L1 Data cache (346.4 FIT in TABLE III) and the 8-way set associative L1 Instruction cache (193.6 FIT in TABLE IV).

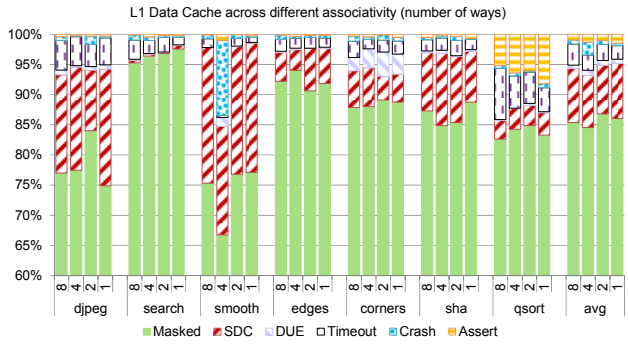

Figure 5: Faults classification in L1 Data cache (associativity).

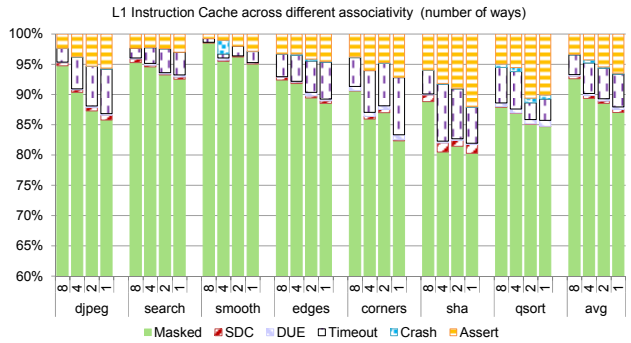

Figure 6: Faults classification in L1 Instr. cache (associativity).

Caches Write Policy

In this part, we study the write back and the write through policies in all caches of memory hierarchy. In Figure 7, the average percentage of masked category in L1 Data cache increases by 6 percentage points when the write through policy is used instead of the write back. In Figure 8, the L1 Instruction cache features almost the same behavior (equivalent percentage of masked category) for both policies since blocks that are evicted by instruction caches are never dirty and thus cannot propagate the fault to lower levels of memory hierarchy. In Figure 9, the write through unified L2 cache has higher masking probability than the write back L2 cache (about 2 percentage points on average). A faulty cache line in L2 Cache that is exclusively allocated to data (not instruction) may propagate the fault to the lower level of memory hierarchy only if a write back policy is used.

In essence, write through caches are more reliable than write back caches in all levels of memory hierarchy[1]. The write through caches feature less FIT than write back caches: 240.4 FIT for L1 Data in TABLE III, 264.0 FIT for L1 Instruction in TABLE IV and 1390.1 FIT for unified L2 in TABLE V.

First Level Cache Prefetchers

In this part, we study the impact of the presence of prefetchers in L1 caches. In Figure 10, the average percentage of masked class of L1 Data cache with prefetcher is very close to the one without prefetcher. In Figure 11, the average percentage of masked class of L1 Instruction cache increases by 5 percentage points when prefetecher is enabled.

In TABLE III, the L1 Data cache with prefetcher (405.4 FIT) and the one without prefetcher (392.3 FIT) have almost the same reliability since their FIT are close. In TABLE IV, the L1 Instruction cache is more reliable with enabled prefetcher because it features less failures in time (278.8 to 413.1 FIT with and without prefetcher respectively). A prefetcher can occasionally reduce the residency time of a cache line (by replacing it) or fill a cache line with useless data that is not used by the processor. Our results show that

[1] In case of write through cache, a transient fault hitting a bit in the cache can only be propagated to the processor when the block is read before its eviction. However, in a write back cache the fault can be propagated to the processor or to the lower levels of memory hierarchy when the block is evicted and contains dirty data.

the presence of a prefetcher enhances only L1 Instruction cache's reliability.

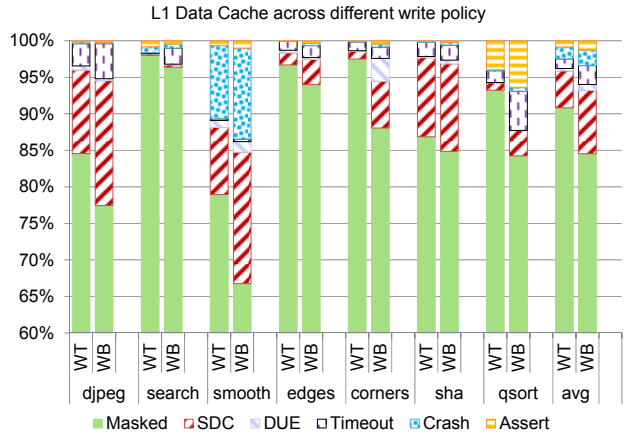

Figure 7: Faults classification in L1 Data cache (write policies).

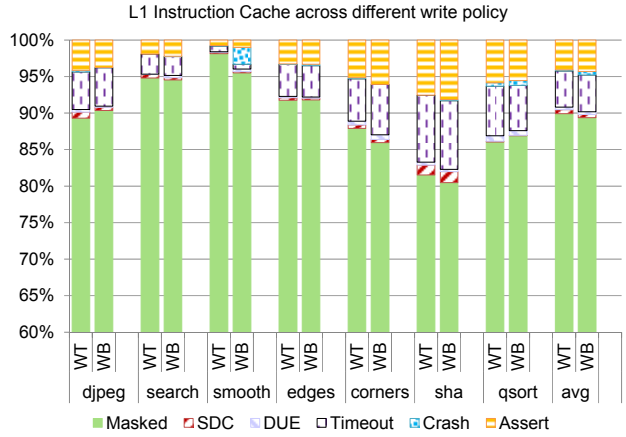

Figure 8: Faults classification in L1 Instr. cache (write policies).

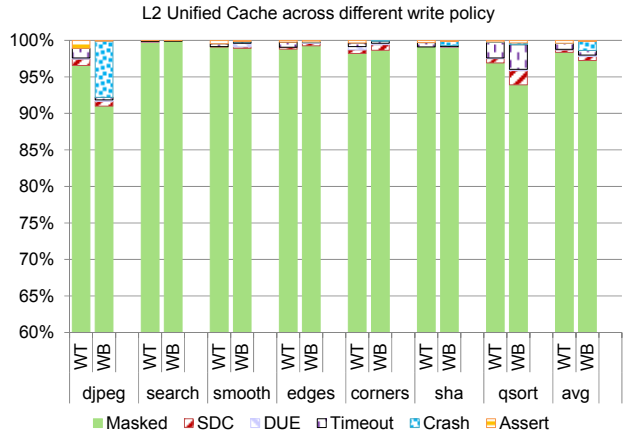

Figure 9: Faults classification in L2 Unified cache (write policies).

b) Function for reliability/performance correlation

In this part, we use the fitness function (defined in Section II.C) to quantify the effect of modifications in microarchitectural parameters both in terms of performance and reliability. Assigning different values to parameter *a,* we adjust the impact of reliability and performance in the design decisions. TABLE III to TABLE VII present the values of the fitness function for all the components of our study and for three different values of parameter *a*: 0.25 (design focused on performance), 0.50 (design balanced between reliability and performance), 0.75 (design focused on reliability). Moreover for each microarchitectural configuration under study, TABLE III to TABLE VII also show the FIT and IPC on average for all benchmarks. Individual benchmarks can be similarly studied. The fitness function values are normalized to the baseline configuration fitness[2]. The "best" fitness values for different design priorities (the three *a* values) are highlighted with shaded cells. Fitness values greater than 1 indicate a design point which improves the fitness compared to the baseline model.

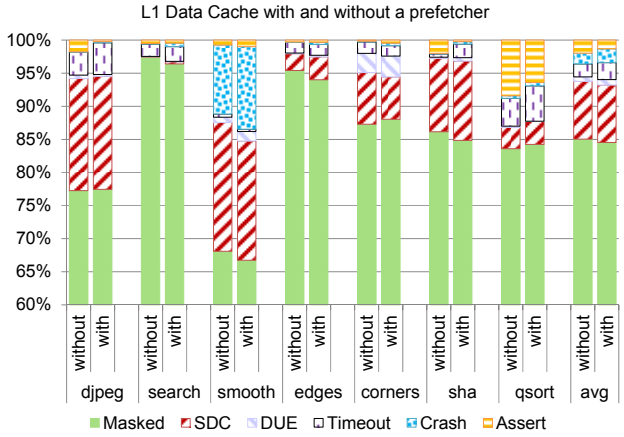

Figure 10: Faults classification in L1 Data cache (prefetcher).

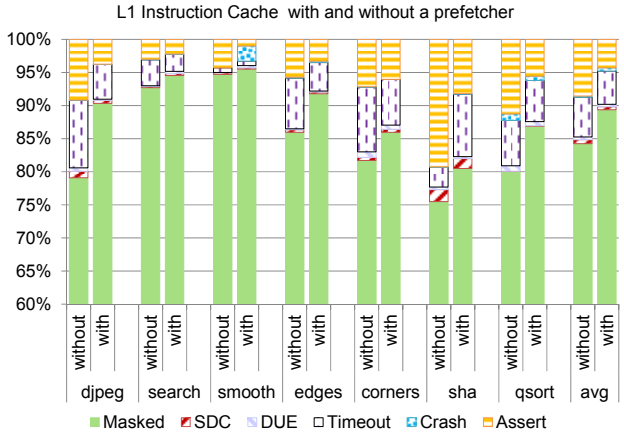

Figure 11: Faults classification in L1 Instr. cache (prefetcher).

IV. CONCLUSIONS

We presented a comprehensive evaluation of the impact of major hardware structures parameters values on the reliability of a microprocessor to transient faults using statistical fault injection on major structures with a state-of-the-art x86-64 fault injector. We also defined a simple and flexible fitness function that aggregates with variable weights the reliability and performance in order to quantify the impact of several design decisions.

[2] For example, if a=0.5, IPC_{base}=0.80, FIT_{base}=100, IPC_{conf}=0.85, FIT_{conf}=120: $fitness_{config}$ = 0.5×1/(120/100) + 0.5×(0.85/0.80) = 0.9479

TABLE III FIT, IPC AND FITNESS VALUES FOR THE L1 DATA CACHE

L1D	Baseline	policy	Associativity			prefetcher	size	
	WB, 4-way prefetcher,32KB	WT	1-way	2-way	8-way	w/o prefetcher	16KB	64KB
FIT	405.4	240.4	366.3	346.4	383.7	392.3	251.1	637.4
IPC	0.7932	0.7606	0.7554	0.7682	0.7541	0.8569	0.7590	0.8043
a	*Fitness*							
0.25	1.000	1.141	0.991	1.019	0.977	1.069	1.121	0.919
0.5	1.000	1.323	1.030	1.069	1.004	1.057	1.286	0.825
0.75	1.000	1.504	1.068	1.120	1.030	1.045	1.450	0.731

TABLE IV FIT, IPC AND FITNESS VALUES FOR THE L1 INSTRUCTION CACHE

L1I	Baseline	policy	Associativity			prefetcher	size	
	WB, 4-way prefetcher,32KB	WT	1-way	2-way	8-way	w/o prefetcher	16KB	64KB
FIT	278.8	264.0	339.7	300.5	193.6	413.1	159.4	427.3
IPC	0.7932	0.7606	0.7554	0.7682	0.7541	0.8569	0.7590	0.8043
a	*Fitness*							
0.25	1.000	0.983	0.919	0.958	1.073	0.979	1.155	0.924
0.5	1.000	1.007	0.887	0.948	1.195	0.878	1.353	0.833
0.75	1.000	1.032	0.854	0.938	1.318	0.776	1.551	0.743

TABLE V FIT, IPC AND FITNESS VALUES FOR THE L2 CACHE

L2	Baseline WB	Policy WT
FIT	2318.9	1390.1
IPC	0.7932	0.7694
a	*Fitness*	
0.25	1.000	1.145
0.5	1.000	1.319
0.75	1.000	1.494

TABLE VI FIT, IPC AND FITNESS VALUES FOR THE INTEGER PHYSICAL REGISTER FILE

Reg. File	baseline	Size	
	256 regs.	64regs.	128regs.
FIT	4.7	4.3	4.6
IPC	0.7932	0.7057	0.8097
a	*Fitness*		
0.25	1.000	0.937	1.023
0.5	1.000	0.985	1.026
0.75	1.000	1.032	1.028

TABLE VII FIT, IPC AND FITNESS VALUES FOR THE LSQ

LSQ	baseline	size	
	32 entries	64 entries	96 entries
FIT	0.5	0.6	0.7
IPC	0.7932	0.7889	0.8025
a	*Fitness*		
0.25	1.000	0.973	0.962
0.5	1.000	0.951	0.912
0.75	1.000	0.929	0.863

ACKNOWLEDGMENTS

This work is supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.

REFERENCES

[1] S.Nassif, N.Mehta, Y.Cao, "A Resilience Roadmap", DATE 2010.
[2] Z.Chishti, A.R.Alameldeen, C.Wilkerson, W.Wu, S.-L.Lu, "Improving Cache Lifetime Reliability at Ultra-low Voltages", MICRO 2009.
[3] R.C.Baumann, "Soft Errors in Advanced Computer Systems", IEEE Design & Test of Computers, vol. 22, no. 3, pp. 258-266, 2005.
[4] V.Sridharan, D.R.Kaeli, "Using Hardware Vulnerability Factors to Enhance AVF Analysis", ISCA 2010.
[5] A.Patel, F.Afram, C.Shunfei, K.Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs", DAC 2011.
[6] N.Foutris, M.Kaliorakis, S.Tselonis, D.Gizopoulos, "Versatile Architecture-Level Fault Injection Framework for Reliability Evaluation", IOLTS 2014.
[7] M.Kaliorakis, S.Tselonis, A.Chatzidimitriou, N.Foutris, D.Gizopoulos, "Differential Fault Injection on Microarchitectural Simulators", IISWC 2015.
[8] A.Biswas, P.Racunas, R.Cheveresan, J.Emer, S.S.Mukherjee, "Computing Architectural Vulnerability Factors for Address-Based Structures", ISCA 2005.
[9] 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.
[10] G.-H.Asadi, V.Sridharan, M.Tahoori, D.Kaeli, "Balancing Performance and Reliability in the Memory Hierarchy", ISPASS 2005.
[11] A.Savino, S.Di Carlo, G.Politano, A.Benso, A.Bosio, G.Di Natale, "Statistical Reliability Estimation of Microprocessor-Based Systems", IEEE Transaction on Computers, vol. 61, no. 11, pp. 1521-1534, 2012.
[12] J.Suh, M.Annavaram, M.Dubois, "MACAU: A Markov Model for Reliability Evaluations of Caches Under Single-bit and Multi-bit Upsets", HPCA 2012.
[13] H.Cho, S.Mirkhani, C.Chen-Yong, J.A.Abraham, S.Mitra, "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design", DATE 2013.
[14] N.George, C.Elks, B.Johnson, J.Lach, "Transient Fault Models and AVF Estimation Revisited", DSN 2010.
[15] M.Kaliorakis, S.Tselonis, A.Chatzidimitriou, D.Gizopoulos, "Accelerated Microarchitectural Fault Injection-Based Reliability Assessment", DFTS 2015.
[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] S.P.Vanderwiel, D.J.Lilja, "Data prefetch mechanisms", ACM Computing Surveys, 2000, vol. 32, no. 2, pp. 174-199.
[19] N.Foutris, D.Gizopoulos, J.Kalamatianos, V.Sridharan, "Assessing the Impact of Hard Faults in Performance Components of Modern Microprocessors", ICCD 2013.
[20] J.Stevens, P.Tschirhart, M.-T.Chang, I.Bhati, P.Enns, J.Greensky, Z.Chisti, S.-L.Lu, B.Jacob, "An Integrated Simulation Infrastructure for the Entire Memory Hierarchy: cache, DRAM, non-volatile memory, and disk", Intel Technology Journal, 2013, vol. 17, no. 1.
[21] M.R.Guthaus, J.S.Ringenberg, D.Ernst, T.M.Austin, T.Mudge, R.B.Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", IISWC 2001.
[22] 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.