Beyond Cross-Section: Spatio-Temporal Reliability Analysis

THIAGO SANTINI, PAOLO RECH, GABRIEL LUCA NAZAR, and
FLÁVIO RECH WAGNER, Federal University of Rio Grande do Sul

A computational system employed in safety-critical applications typically has reliability as a primary concern. Thus, the designer focuses on minimizing the device radiation-sensitive area, often leading to performance degradation. In this article, we present a mathematical model to evaluate system reliability in spatial (i.e., radiation-sensitive area) and temporal (i.e., performance) terms and prove that minimizing radiation-sensitive area does not necessarily maximize application reliability. To support our claim, we present an empirical counterexample where application reliability is improved even if the radiation-sensitive area of the device is increased. An extensive radiation test campaign using a 28*nm* commercial-off-the-shelf ARM-based SoC was conducted, and experimental results demonstrate that, while executing the considered application at military aircraft altitude, the probability of executing a two-year mission workload without failures is increased by 5.85% if L1 caches are enabled (thus increasing the radiation-sensitive area) when compared to no cache level being enabled. However, if both L1 and L2 caches are enabled, the probability is decreased by 31.59%.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; B.8 [**Hardware**]: Performance and Reliability—*Reliability, testing and fault-tolerance*

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: Cache memories, embedded systems, performance, reliability

ACM Reference Format:
Thiago Santini, Paolo Rech, Gabriel Luca Nazar, and Flávio Rech Wagner. 2015. Beyond cross-section: Spatio-temporal reliability analysis. ACM Trans. Embed. Comput. Syst. 15, 1, Article 3 (December 2015), 16 pages.
DOI: http://dx.doi.org/10.1145/2794148

1. INTRODUCTION

Safety-critical and aerospace applications like biomedical implantable devices, automotive control systems, and plane or satellites stabilizers and control circuitry have peculiar requirements that differ from commercial ones. The main difference is that, when a safety-critical or space system is designed, performance is often sacrificed in order to enhance reliability or reduce power consumption, whereas on a typical project, the goal is to increase performance without exceeding power and, at times, reliability specifications. This difference has led designers to produce specific hardened chips to meet reliability requirements in safety-critical applications. Nevertheless, lately, the possibility arises of employing Commercial-Off-the-Shelf (COTS) systems in applications in which error occurrences must be minimized. The main reason for such a choice is that hardened devices are typically very expensive because they

Authors' addresses: P. Rech, G. L. Nazar, and F. R. Wagner, Institute of Informatics, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500, Bloco IV, 91501-970, Porto Alegre, Brazil; emails: {prech, glnazar, flavio}@inf.ufrgs.br.
Author's current address: T. Santini, Wilhelm-Schickard-Institute for Computer Science, University of Tübingen, Sand 14, C206, 72076, Tübingen, Germany; email: thiago.santini@uni-tuebingen.de.

DOI: http://dx.doi.org/10.1145/2794148

require unique circuit design and lithography, and the volumes of produced devices are very low. COTS components, on the contrary, are low cost, flexible, and have fast time-to-market and low power consumption. Such devices are already being used in particular applications, such as the on-board computer in NASA's PhoneSat project—a nanosatellite built around COTS smartphones running the Android operating system [NASA 2014]. When reliability is a major concern, the use of general-purpose devices must be carefully evaluated. As technology scales down, CMOS devices are becoming more susceptible to soft errors induced by ionizing particles; in fact, nowadays, radiation-induced failures are a concern not only in radiation-harsh environments such as space, but also in milder environments, such as at sea level. High-energy neutrons generated by the interaction of cosmic rays with the terrestrial atmosphere may in fact have enough energy to corrupt data stored in SRAM memories or affect logic computations [Baumann 2005; Ziegler et al. 1996].

The *cross-section*, defined as the radiation-sensitive area of a device, is commonly selected as the metric to evaluate the device's or system's radiation sensitivity. A larger cross-section means that it is more likely for a neutron hitting the device to generate a failure. To ensure higher reliability in a system, the configuration that provides the smallest cross-section is typically chosen. As an example, resources like caches can be disabled to reduce their cross-section [Lesea et al. 2014; Rebaudengo et al. 2003]. However, extra resources like caches, although not necessary for computation, have a critical impact on performance. Forcing the system to work without these resources will result in longer execution time. Even if performance is not a primary concern in safety-critical applications, having a longer execution time allows the system to be exposed for a longer time to radiation before completing execution. Consequently, the impact on system reliability is twofold: On one hand, reducing the cross-section reduces the probability of one impinging particle hitting a sensitive part of the device, thus decreasing the probability of a failure occurring; on the other hand, increasing the exposure time increases the number of particles that hit the device during execution, thus increasing the probability of failure occurrence. Then, depending on how these effects balance each other out, system reliability may be positively or negatively impacted or even remain unaltered.

In this article, we present a novel approach to evaluate system reliability that goes beyond the cross-section by considering the temporal impact of enabling extra resources. Our method ties the device radiation sensitivity and the application performance (i.e., execution time) together through the well-established Poisson model.

Additionally, we experimentally refute the hypothesis that reducing the cross-section always increases the reliability of computing systems by presenting a counterexample. We conducted an extensive neutron test campaign on state-of-the-art COTS-embedded ARM processors using cache memories as extra resources. The experimental results demonstrate that, for the system under test, disabling all caches reduces the cross-section by a factor of 28.2 when compared to all caches enabled and by 4.49 when compared to only the L1 caches enabled. Nevertheless, the application has a higher probability of completing the execution correctly when the L1 caches are enabled, although this probability decreases when both L1 and L2 caches are enabled.

The remainder of this article is organized as follows: Section 2 presents a background on electronic device radiation sensitivity, focusing on cache memories; Section 3 describes the mathematical model used to evaluate the realistic reliability of a device executing a given code; Section 4 presents the radiation experiment setup and gathered results; and Section 5 draws main conclusions and presents future work.

2. BACKGROUND

Radiation-induced errors are a major dependability threat for state-of-the-art electronic devices due to reduced operating voltages and capacitances that, in turn, lead

to a reduced critical charge. As a result, Single Event Effects (SEEs) become more frequent even at ground levels [Dixit and Wood 2011; Baumann 2005; Ziegler et al. 1996]. Most notably, transient effects such as Single Event Transients (SETs) affecting combinational logic and Single Event Upsets (SEUs) affecting memory elements have a high probability of interfering with circuit operation [Baumann 2005].

2.1. Radiation Sensitivity

The cross-section is the most widely used metric to evaluate the sensitivity to radiation of a device. By definition the cross-section, usually expressed in cm^2, is the sensitive area of the device; that is, the area that, if hit by an impinging particle, generates a failure [Baumann 2005]. The larger the cross-section, the more likely is a radiation-induced failure. The cross-section is obtained experimentally by dividing the number of observed errors by the total particle fluence (i.e., the number of particles hitting the device per unit area) [JEDEC 2007]. Normally, particle accelerators, neutron beams, or radioactive sources are employed to experimentally evaluate the device cross-section because they provide a controlled and typically high flux, whereas real tests in the field are usually very expensive and time-consuming [Ziegler et al. 1996]. To evaluate the error rate of the device, it is then sufficient to multiply its cross-section by the expected particle flux in the environment where it will be deployed.

The reliability of processors is normally related to the execution of a given algorithm, benchmark, or workload that should be as representative as possible of the application that the processor is going to perform. The system cross-section σ can then be evaluated as

$$\sigma = \frac{\lambda}{\phi} \tag{1}$$

where λ is the observed output error rate (i.e., errors per unit time) and ϕ is the particle flux (i.e., number of particles hitting the device per unit area and unit time).

Because the fluence is expressed in $particles/cm^2$, the resulting cross-section has the dimension of an area. However, it is not a strictly geometrical measure because it depends not only on the physical size of the employed resources, but also on their sensitivity to radiation and their criticality to the correct execution of the application. As such, it expresses the probability of having one impinging neutron corrupting the processor while executing the given code in such a way that an error appears in the output. The processor cross-section, in other words, gives an indication of the sensitivity of the resources required by the code to be executed.

2.2. Cache Memories and Radiation

Cache memories are considered the most vulnerable parts of modern computing systems [Manoochehri et al. 2011; Mukherjee et al. 2004; Liden et al. 1994] since they occupy about 60% of the on-chip area in today's microprocessors. Moreover, to be fast and efficient, cache memory cells are built as small as possible and, thus, their capacitance and critical charge are lowered, hence increasing the probability of having a memory cell corrupted by ionizing radiation. Protection techniques that may be applied to cache memories, including parity and ECC, incur in extra area and significant additional power consumption and even lead to performance degradation [Asadi et al. 2005]. In this scenario, the probability of having a Multiple Bit Upset (i.e., one impinging neutron corrupting more than one memory cell) is increasing with the shrinking of the technology node [Ibe et al. 2010; Maiz et al. 2003]. Moreover, caches are typically compact and dense; thus, the memory cells are close to each other, exacerbating the possibility of having one single impinging particle interacting with more than one transistor. The ECC design is then becoming more and more challenging. Although it has been shown that Multiple Bit Upsets could be mitigated through the use of memory

interleaving [Baeg et al. 2009], there is no guarantee that such hardware support (or even ECC) will be available in COTS embedded systems.

Cache memory reliability has been investigated in previous works. In Rebaudengo et al. [2003], the authors injected SEUs into the pipeline registers, register file, instruction cache, and data cache by instrumenting the HDL code of the Leon core [Gaisler 2014]. For a given application, the number of faults injected is proportional to the time required to run the application. Overall, the failure rate of four applications is measured. A comparison is made between the Leon core with its L1 caches enabled and disabled, showing that enabling the L1 caches increased the failure rate by a factor ranging from 5.46 to 24.70. In Asadi et al. [2005] and Asadi et al. [2006], the sensitivity of different cache configurations is shown to be very diverse based on their vulnerability factors as experimentally obtained with an extended version of the SimpleScalar simulator [Burger and Austin 1997], but their relation to the cores vulnerability is not evaluated. Cache size impact on time-constrained systems is analyzed in Cai et al. [2006]; the authors show the existence of an optimal or Pareto-optimal cache size when optimizing for energy, performance, and reliability. The evaluation was realized on MPARM [Benini et al. 2005], a cycle-accurate simulator.

A straightforward solution employed to reduce the occurrence of radiation-induced failures is to minimize the system sensitive area, which is the amount of employed resources that are prone to be corrupted. As mentioned, cache memories are addressed as one of the most critical resources in a microprocessor. These memories are employed to reduce execution time by reducing memory access latencies and thus are labeled as unnecessary in a safety-critical design based on COTS components and disabled [Rebaudengo et al. 2003].

Nevertheless, this engineering solution is worth only assuming that disabling caches will indeed improve reliability in spite of the increased execution time. In fact, even if performance is not a major concern in safety-critical applications, it must be considered that, from a radiation reliability perspective, slower execution turns into longer exposure time. In other words, disabling caches will result in higher reliability only if the increased exposure time due to slower performances is compensated by the decrease in sensitive area. Additionally, due to the huge difference in improvements between processor performance and memory access time, the gap between logic and memory performance becomes wider at each new device generation [Lu et al. 2012]. The performance ratio between executing a task with and without caches then increases. As demonstrated in Section 4 by a counterexample, in modern devices, the assumption that the increased exposure time will always be compensated by the decrease in sensitive area no longer holds. As a result, in certain particular conditions, enabling caches may not only preserve performance but also increase the overall system radiation reliability. It is important to note that, even in these particular conditions, enabling caches will nonetheless increase the rate of errors per unit of time; however, the rate of errors per task execution will be diminished.

3. APPLICATION RELIABILITY

The most widely used method to increase reliability is to reduce the processor cross-section by disabling or reducing resources that are not essential for computation, like caches. Nevertheless, as stated in the previous section, the execution time of the code varies consistently when processor resources are changed (caches, for instance, were introduced specifically to reduce memory access latency and increase the performance of processors). On a realistic application, the exposure time of the code will be modified when the available resources are limited by having the code disposing of the higher amount of resources exposed for a much shorter time with respect to others. As a result, enabling extra resources increases the probability of having one impinging

particle generate a failure (i.e., the cross-section) but reduces the number of particles that hit the device during computation. The shorter exposure time may then compensate for the larger exposed area.

The basic concept explored in this work is that the probability P of executing a given application correctly is inversely related not only to device sensitivity, expressed by its cross-section σ and by the average particle flux to which the device is exposed ϕ, but also by the total exposure time t, which is determined by the time the device requires to execute the application. Hence, assuming designers cannot modify the flux to which the system is exposed, minimizing both sensitivity and execution time are the two main approaches to increase P. However, these two approaches are often conflicting since the most common mechanisms to improve t have a negative impact on σ.

As an example, a common practice to reduce σ is to disable caches on a processor. Nevertheless, memory hierarchies also play a key role in performance. Thus, even if σ is significantly reduced, t is going to increase, and no assumption on P could be given a priori.

A second example of the tradeoff between σ and t is given by parallel processors. Exploiting parallelism naturally introduces additional sensitive area due to the very definition of parallel execution. However, the code execution time will benefit from parallelism, potentially increasing P [Rech et al. 2014].

Similarly, increasing the processor frequency may increase its cross-section. In fact, higher clock frequencies increase the sensitivity to SETs [Baumann 2005] since they increase the probability of sampling glitches generated by impinging particles. Nonetheless, a higher frequency allows the processor to reduce the execution time t and thus may increase P.

Additionally, even algorithm implementation choice presents a similar tradeoff. For example, the use of lookup tables speeds up execution at the cost of an increase in memory usage. Although t is reduced, more memory used means more memory cells prone to corruption and thus a larger σ.

In order to compare different solutions in terms of their reliability, an appropriate metric to evaluate their operative reliability taking both σ and t into account is required. Thus, we propose a model to quantify the probability of a given solution executing successfully to be used as a measure of its operative reliability.

3.1. Application Reliability Model

Radiation-induced faults are stochastic-independent events. These faults are also rare when compared to the completion of a program run (i.e., the mean time between failures is much larger than the time it takes to execute the program once). Therefore, it is reasonable to assume radiation-induced errors distributed over time following a Poisson distribution. The probability P of having k failures during the execution time t can then be expressed as

$$P(k, \lambda, t) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}, \tag{2}$$

where λ is the application failure rate.

In this work, we are interested in finding the probability of executing the code successfully (i.e., without failures). Consequently,

$$k = 0, \tag{3}$$

and the probability of executing the application successfully is

$$P(0, \lambda, t) = \frac{e^{-\lambda t}(\lambda t)^0}{0!} = e^{-\lambda t}. \tag{4}$$

While the execution time t can be directly measured from the running application, the failure rate λ can be inferred from Equation (1) as

$$\lambda = \phi\sigma. \tag{5}$$

As a result, we can evaluate the probability for a device to execute the desired application correctly while exposed to any given flux through

$$P(\phi, \sigma, t) = e^{-\phi\sigma t}, \tag{6}$$

which can then be used as a metric of operative reliability.

3.2. Area-Performance Tradeoff

The proposed analysis can be extended to consider not only the operative reliability, but also the impact of the area-performance tradeoff on application reliability. For this purpose, consider two candidate solutions (a and b) for the same problem, where solution a increases σ but decreases t, and solution b decreases σ but increases t.

Let σ_a and t_a denote, respectively, the cross-section and execution time for solution a, and σ_b and t_b the same variables for solution b. When comparing these two solutions in terms of reliability, we can evaluate if solution a has higher P than solution b when exposed to a given flux ϕ:

$$P(\phi, \sigma_a, t_a) > P(\phi, \sigma_b, t_b). \tag{7}$$

This can be reduced to

$$e^{-\phi\sigma_a t_a} > e^{-\phi\sigma_b t_b} \tag{8}$$

$$\phi\sigma_a t_a < \phi\sigma_b t_b \tag{9}$$

$$\frac{\sigma_a}{\sigma_b} < \frac{t_b}{t_a}. \tag{10}$$

Therefore, as shown in Equation (10), solution a will have a higher operative reliability when the observed increase in sensitive area brought by a, in comparison to b (the left-hand side of Equation (10)), is lower than the speed-up provided by a relative to b (the right-hand side).

Note that we assume that both solutions are exposed to the same flux, which is a reasonable assumption since they are to be deployed in the same environment. Consequently, Equation (10) is flux-independent and allows us to assess which solution has a better operative reliability based solely on properties of the platform and on the speed-up provided.

4. CASE STUDY

We performed an extensive neutron test campaign on embedded ARM processors to illustrate a counterexample that confutes the hypothesis that a lower cross-section always improves reliability. Moreover, we take advantage of experimental results to provide a practical application of the proposed probabilistic model in analyzing the impact of enabling extra on-chip resources on the effective reliability of the application.

4.1. Device Under Test

The Device Under Test (DUT) is the Xilinx Zynq™-7000 AP SoC implemented in a 28nm CMOS technology. The DUT is mounted on the Zedboard development kit and disposes of 2 ARM®Cortex™-A9 cores with a maximum frequency of 667MHz. Each core has 32KB Level 1 4-way set-associative instruction and data caches, and they share a 512KB 8-way set-associative Level 2 cache [Digilent 2014]. Both cache levels have parity support, which was disabled during our experiments; assuming single-bit

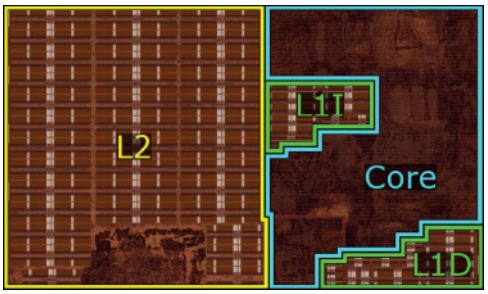

Fig. 1. Approximate area of essential resources for computation (Core) and extra resources used to speed-up execution (L1 and L2 Caches) marked on a polysilicon die photo of a Cortex-A9 with 32KB Instruction/Data L1 Caches and 512KB L2 Cache.

corruptions and that detected faults are observable errors, enabling parity support would increase the observable error rate due to the detection of benign faults (i.e., faults that would otherwise not impact application execution would raise exceptions if parity was enabled). During experiments, only one of the two available cores is used, and the resulting system is similar to that of Figure 1.

The multiple cache levels available in state-of-the-art embedded processors introduce different options for designers. For the DUT, which has L1 and L2 caches, it is reasonable to assume that there is a given configuration, such as using no caches, only L1 caches, or both cache levels, that optimizes reliability (in terms of exposure time and sensitive area). It is not obvious, however, which is the optimum choice. The optimal configuration depends on the sensitive area introduced by each enabled level and on the speed-up provided by each of them as well, which are functions of both application and platform properties. Making an informed choice for the platform and application at hand can significantly improve reliability, as shown in the remainder of this article.

4.2. Application and Tested Cache Configurations

A single specific instance in which reliability is improved even if the sensitive area is increased suffices as a counterexample. Furthermore, no requirement is given on the complexity of the application. Therefore, based on these criteria and the fact that beam time is limited, we opted to use a single application during our experiments since this allows us to minimize the statistical error of our results.

To experimentally measure how the DUT reliability varies as extra resources are enabled, we generated the same application in three versions: (i) both L1 and L2 caches enabled, (ii) only L1 caches enabled, and (iii) all cache levels disabled. Only one of the two available cores is allowed to run to avoid cache access conflicts since the L2 cache is shared. This core then disables parity at the L2 cache, sets a dummy function as the L1 parity interruption handler, and sets the cache hierarchy configuration chosen at compilation time. After the initial setup, a banner is printed and the application executes repetitively. The banner is used to detect reboots and make sure that the correct cache configuration is being used.

Matrix multiplication was selected as benchmark since it is a common workload in critical embedded systems used, for example, in control and filter operations. The detection of incorrect answers is achieved by comparing the computed results of multiplying input matrices A and B with a golden copy G.

To increase all caches' utilization, we designed a code in which 200 multiplications of 25×25 integer matrices ($25 \times 25 \times 4 = 2500$ Bytes/Matrix) are executed. Each multiplication has its own set of input matrices A, B, and golden copy G ($2500 \times 3 = 7500$ Bytes/Set), amounting to at least $200 \times 7500 \approx 1.43$ MB of data. As shown in

Algorithm 1, after DUT initialization, the application starts: All matrices are defined as local variables in the stack, and, for each of the 200 sets of input matrices, A_i is multiplied by B_i; the resulting matrix is then compared to the expected result G_i, and, if they differ, a failure flag is raised. After the 200 matrix multiplications are completed, errors are reported and a new application execution is triggered.

ALGORITHM 1: Application Under Test

```
setup_caches();
print_banner();
while True do
    // Applications Start
    Fail ← False;
    for i ← 1 to 200 do // Unrolled
        init_in_stack(A_i, B_i, G_i);
    end
    for i ← 1 to 200 do // Unrolled
        C ← A_i * B_i;
        if C ≠ G_i then
            Fail ← True;
        end
    end
    print(Fail);
    reset_stack_pointer();
    // Application End
end
```

To evaluate the approximate cache occupancy (i.e., number of valid blocks per total number of blocks) during application execution, we used a modified version of the gem5 simulator [Binkert et al. 2011] using a similar configuration to the DUT. Figure 2 shows the resulting instantaneous cache occupancy for the L1 instruction, L1 data, and L2 caches. After a reboot, the cache occupancy is zero due to the cold start. During the first execution, the L1 data cache is quickly filled up due to data initialization and remains at 100% occupancy for the rest of the time. The L2 cache reaches close to 98% occupancy during data initialization, reaching 100% occupancy quickly afterward. The L1 instruction cache is filled during the whole period of application execution, signaling that different instructions are used throughout the execution, saturating around 92.97% occupancy. These cold executions represent only an insignificant amount ($<$0.03%) of the total executions. During subsequent executions, the L1 data and instruction caches remain saturated at around 100% and 93%, respectively, while the L2 cache has a few lines invalidated at the beginning of the execution and quickly reaches 100% occupancy again.

Table I shows the average cache occupancy and the time it takes to execute the proposed benchmark once for each of the three cache hierarchy configurations. As expected, when both L1 and L2 caches are enabled, the performance of the DUT is higher, being almost 8 times faster than the version in which all caches are disabled. When only the L2 cache is disabled, but L1 caches remain enabled, the code latency is increased by just 10%.

As stated in the previous section, caches are one of the resources with the highest error rate. The exposed area of the DUT when both L1 and L2 are enabled is larger than the area when no cache is used. Moreover, having a longer execution time increases the exposure time of data stored in internal registers. While data are fetched from main memory, registers remain exposed to radiation, and the data held within those registers

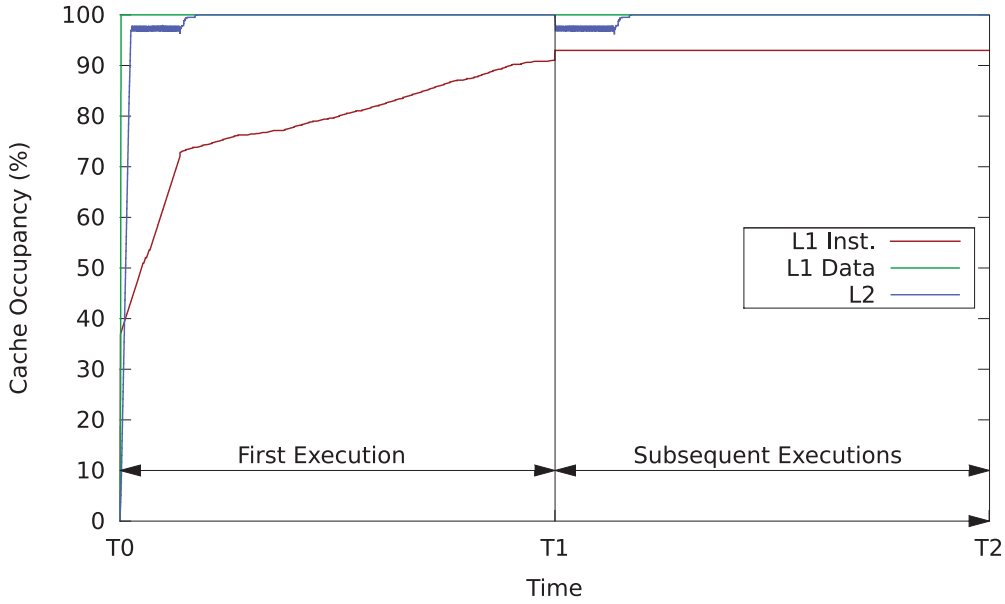

Fig. 2. Instantaneous cache occupancy for the first and subsequent executions.

Table I. Application Performance with Different Cache Configurations

Version	Execution Time (s)	Avg. L1 Inst. Occupancy(%)	Avg. L1 Data Occupancy (%)	Avg. L2 Occupancy (%)
No Cache	5.04E-01	N/A	N/A	N/A
L1	7.10E-02	92.97	100	N/A
L1 + L2	6.40E-02	92.97	100	99.57

are likely to be used for further computations. Thus, a failure in the register holding data while the processor is waiting because of memory latency is likely to propagate to the output, generating an output error. Remaining resources, such as portions of the speculative state, may also vary (although in a less predictable manner) due to the execution of more instructions in parallel. Regarding logic, it is worth noticing that in this particular case the application speed-up is not achieved by increasing the processor frequency, which would increase SET occurrences [Baumann 2005]. The frequency of operation of the DUT is constant, and the longer execution time observed when L2 or both L1 and L2 are disabled is caused by latency for gathering inputs from the main memory.

4.3. Experimental Setup

Radiation experiments were performed at Los Alamos National Laboratory (LANL) Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House II, called ICE House II. The ICE House II beam line provides a white neutron source that emulates the energy spectrum of the atmospheric neutron flux. The available neutron flux was approximately $1 \times 10^6\, n/(cm^2 s)$ for energies above 10 MeV. The beam was focused on a spot with a diameter of 2 inches, which provided uniform irradiation of the System on a Chip (SoC) without directly affecting nearby board power control

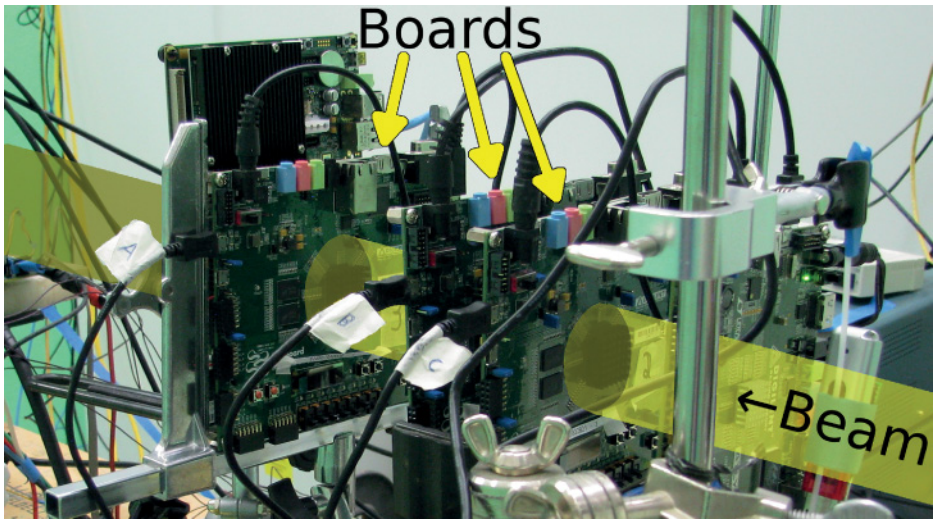

Fig. 3. Experimental setup mounted at LANSCE ICE House II.

circuitry. The DRAM chips were located outside the beam; thus, we assume no errors are generated within them.

The overall SoC silicon die was then irradiated, thereby taking into account the sensitivities of the core and caches. Furthermore, the individual sensitivities of different components of the system are automatically and implicitly taken into account by the experiment. Since the memory cells used in register files, pipelines, and caches are likely to have different dimensioning, their cross-section is unlikely to be the same. The conducted experiments accurately capture these subtle differences to provide reliability measurements.

To reduce uncertainty in the experimental results, we irradiate in parallel three DUTs (see Figure 3), running in each the same application but with a specific cache hierarchy configuration: (i) All caches disabled, (ii) only L1 enabled, and (iii) L1 and L2 enabled.

Irradiation was performed with normal incidence; experiments were conducted at a constant room temperature, and the boards were powered with a nominal voltage of 12V. Three boards with the same hardware revision were aligned with the beam, placed at 61, 62, and 65 inches from the source, respectively. A beam flux derating factor was calculated for each board, to take into account the beam degradation due to distance from the source. To minimize the statistical error and avoid experimental results bias on the selected board and distance derating factor, the codes with the three different cache hierarchy configurations running the aforementioned application were executed alternatively in all three devices. Each version was executed for more than 80 hours under the beam, receiving a total fluence of $2 \times 10^{11}\, n/cm^2$. As at sea level, the natural neutron flux is estimated to be about $13\, n/(cm^2 h)$ [JEDEC 2006]; during our experiments, the boards received the equivalent to 1.7×10^6 years of exposure in the natural environment. It is worth noticing that even if the flux of neutrons in LANSCE is several orders of magnitude higher than the natural one, the application workload and runtime were tuned to make negligible the probability of having more than one neutron generating a failure in one single code execution (estimated to be no higher than 1.41×10^{-7} through Equation (2) with $k > 1$). This allows the scaling of experimental data in the natural radioactive environment without introducing artificial behaviors.

Table II. Result Summary

Version	Correct Execution	Silent Error	Functional Interruption	Experiment Duration (days)
No Caches	7.25E5	35	13	4.29
L1	5.40E6	221	35	4.5
L1 + L2	4.51E6	1092	38	3.56

Table III. Cross-Sections (cm^2)

Version	Silent Error	Func. Interruption	Overall
No Cache	(2.22±0.18)E-10	(7.22±0.58)E-11	(2.94±0.24)E-10
L1	(1.15±0.09)E-09	(1.82±0.15)E-10	(1.33±0.11)E-09
L1 + L2	(8.03±0.64)E-09	(2.76±0.22)E-10	(8.31±0.66)E-09

4.4. Experimental Results

Each benchmark execution was classified as follows:

Correct. The application produced the expected output of a fault-free environment. No error was detected when compared to the golden copy.

Silent Error. The application produced a different output than that of a fault-free environment. This includes errors detected by comparing the output to the golden copy, irrecoverable situations, and whenever garbage is found in the output (i.e., the UART or its flow control were corrupted).

Functional Interruption. A reboot was detected without an explicit reboot order from the host PC.

Table II summarizes the obtained experimental results, indicating the number of correct executions, silent errors, functional interruptions, and the duration of the experiment. We attributed the same weight to silent errors and functional interruptions when calculating the overall DUT cross-section. Table III shows the resulting cross-section for silent errors, functional interruptions, and the overall DUT cross-section when executing the benchmark for each cache hierarchy configuration. The values are shown with relative intervals to account for neutron count uncertainty. The cross-section is evaluated by dividing the experimentally observed error rate by the flux, representing then the probability for an impinging neutron to generate an observable failure (i.e., the cross-section indicates the sensitive area for each configuration).

As shown in Figure 4, if just the cross-section is used as a metric to evaluate the code to be used in a safety-critical application, the version without caches will be pronounced the more reliable since it has a smaller cross-section. Nevertheless, the execution time of the codes must be taken into account because a longer execution time increases the number of particles hitting the device, potentially undermining the benefit in terms of reliability that a reduced cross-section brings.

Figure 5 shows the probability of executing without failures according to Equation (6) in terms of consecutive executions when exposed to the approximated neutron flux at military aircraft altitude (Altitude $\approx$ 60,000 ft; Flux $\approx 4680\, n/(cm^2 h)$ [Quinn and Graham 2005]). As a reference, we consider DARPA's Vulture program, whose goal is to enable an Unmanned Aerial Vehicle (UAV) capable of operating uninterrupted for more than five years at altitudes higher than 60,000 ft [DARPA 2014]. A small fleet of 10 such UAVs during a two-year mission translates to a workload of roughly 1.25 billion executions of the studied application with caches disabled. Our results show that enabling only the L1 caches increases the probability of completing the assigned workload without failures by 5.85% when compared to the configuration in which all caches are disabled (from 0.831 to 0.889), while enabling both L1 and L2

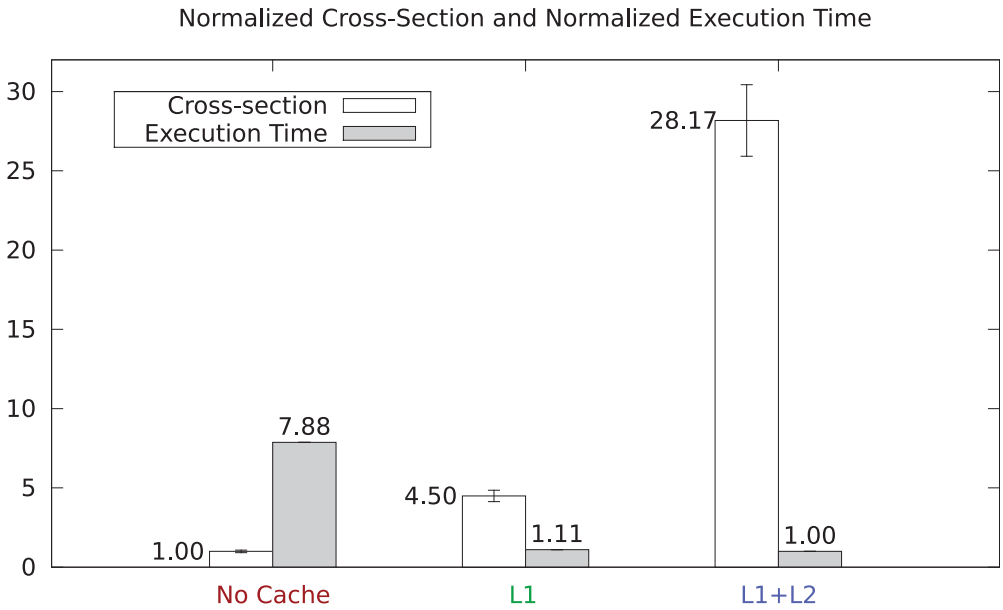

Fig. 4. Impact of different cache hierarchies on the cross-section and execution time of the application under test.

Fig. 5. Probability of multiple executions without failures at military aircraft altitude ($\phi = 4680n/(cm^2h)$).

Table IV. Comparison Table

a vs b	σ_b/σ_a	t_a/t_b	Smaller Cross-Section	More Reliable
No Cache vs L1	4.50	7.15	No Cache	L1
No Cache vs L1+L2	28.17	7.88	No Cache	No Cache
L1 vs L1+L2	6.28	1.1	L1	L1

caches actually reduces it by 31.59% (from 0.831 to 0.515). Thus, the highest reliability is achieved when L1 is enabled, which is not the configuration with the lowest cross-section. It is worth noting that as the number of consecutive executions increases so does the reliability gap between the considered configurations, with the configuration with only the L1 cache enabled being the more reliable.

Furthermore, one may claim that if the application were executed during a fixed period of time (e.g., for two years), the version without caches would yield fewer failures. This is correct since that is the version with the smallest cross-section and thus the smallest failure rate. Nevertheless, this is an unfair comparison reliability-wise because the workload executed by the version without caches is much smaller than the others. In this case, one could take advantage of the faster execution time to apply fault tolerance techniques, such as a time modular redundancy scheme, to reduce the application failure rate. For instance, in this case study, one could multiplex up to seven application executions with only the L1 cache enabled during the time it takes to execute a single time in the version with caches disabled. A software-implemented voter could then use the output of these seven executions to generate a single output, significantly reducing the amount of silent errors. The amount of functional interruptions would most likely remain the same because the system would become unresponsive before even reaching the voter, but these are typically a minority of the observable errors because only specific corruptions lead to them. A similar case may be made for real-time systems in which the system has a certain periodic workload to be processed and a time limit to produce valid outputs; note that the deadline for the workload can not be shorter than the time it takes for the slowest system version to execute. What our model shows is that a system with a slightly larger cross-section but that is much faster can produce this valid output in a fraction of the available time and with reduced exposure. During the slack (i.e., until the next input data arrive), the system is idle and can perform reliability-oriented operations, such as flushing or reloading the contents of the caches. If enough time is available, the system can even be completely rebooted. As a result, the observable failure rate is lower even though the fault rate is higher, thus positively affecting system availability.

Table IV allows one to easily compare the different configurations using Equation (10). The second column shows the increase observed in the cross-section, whereas the third column shows the speed-up provided by the second configuration of each table row. The fifth column indicates the reliability conclusion based on Equation (10).

The basic reading is that, as described in Section 3.2, whenever the speed-up (t_a/t_b) is higher than the increase in the cross-section (σ_b/σ_a), configuration b has a higher probability of completing a given workload correctly. Thus, it can be concluded that, in this case, enabling the L1 cache not only improves performance but also increases reliability to radiation-induced errors, contrary to a conclusion based solely on cross-sections. In cases where the speed-up is not higher than the increase in the cross-section, the latter becomes the dominant factor, and, thus, the cross-section predicts adequately the more reliable version (e.g., the last two rows of Table IV).

It is worth noting that enabling both cache levels does not result in the same effect. Typically, the first cache level is smaller and focuses on minimizing hit time, whereas the secondary cache level tends to be much larger and focuses on reducing miss rates [Patterson and Hennessy 2013]. Furthermore, since the primary cache filters accesses

with good locality, the local miss rate of the secondary cache is much higher than the global miss rate. The first implication of this difference between the cache levels is on speed-up provided: Intuitively, the speed-up provided by the primary cache (relative to the system with no caches) is likely to be much larger than the speed-up provided by the secondary cache (relative to the system with only a primary cache). The second implication of this difference is on the sensitive area incurred by each level: Since smaller caches are less vulnerable than larger caches [Asadi et al. 2005], the increase in sensitive area due to the secondary cache level is larger than the increase in sensitive area due to the primary cache. Therefore, the primary cache level provides a much higher speed-up per increase in sensitive area ratio than the secondary cache level, making it more likely to increase reliability than the secondary cache level. As a result, the L2 cache is much less likely to improve reliability since it provides a small speed-up while significantly increasing the cross-section. This shows that, for a state-of-the-art processor, enabling the L2 cache will need a huge speed-up, which may not be attainable, to achieve the same reliability benefit obtained by enabling the L1 caches; in this case study, the speed-up should be higher than 6.28 times.

Please note that these results cannot be generalized a priori (i.e., enabling the L1 caches does not necessarily improve reliability for all applications). Although enabling extra resources is likely to increase performance and cross-section, quantifying these increases in an analytical manner is a complex task that depends on several factors (e.g., how the application uses these extra resources, how sensitive these resources are). Thus, these increases should be evaluated on a case-by-case basis and taken into account using the method proposed in this work.

5. CONCLUSION AND FUTURE WORK

In this article, we presented a generic model to evaluate the realistic reliability of a computing system. The proposed model takes into account both cross-section and exposure time and can be used to evaluate the best configuration in terms of reliability for an application given a set of solutions exploring the area-performance tradeoff.

A case study with a state-of-the-art processor was presented and evaluated and shows that, in this particular case, the best choice in terms of reliability for the chosen application is to enable only the L1 caches. This case study serves as a clear counterexample to the assumption that the reduction of the radiation-sensitive area always increases reliability, which seems to be prevalent in the community (e.g., focus on cross-section/failure rate/FIT/MTBF as equivalent to reliability and dismiss the resulting execution time overhead as a less important penalty). The performed evaluation demonstrates that the cross-section alone does not yield a proper measure of reliability for an embedded application. It is fundamental to also take the exposure time into account to have a precise indication of system reliability. Moreover, we showed on the basis of this counterexample that there is no go-to solution when it comes to reliability, such as always disabling or always enabling caches. Tuning the spatiotemporal tradeoff on an application or application profile basis is required when reliability is at stake.

Future work includes modeling and analyzing common cases where this tradeoff is found, such as those mentioned in Section 3, to provide valuable information and guidelines for designers. Additionally, we are also interested in applying our model to evaluate Software Implemented Fault Tolerance (SWIFT) techniques, such as EDDA [Oh et al. 2002] and ACCE [Vemu et al. 2007]. Finally, it is also worth investigating different classes of applications and whether they exhibit similar tradeoff behaviors, although these results may not be generalizable to other systems due to factors that are external to the applications (e.g., processor architecture, memory organization, operating system, technology node size, and size of input data).

ACKNOWLEDGMENTS

Authors would like to thank Heather Quinn, Thomas Fairbanks, Steve Wender, and Tanya Herrera from Los Alamos National Laboratory, Los Alamos, NM, for their help in scheduling beam time in LANSCE and for the support during experiments.

REFERENCES

G.-H. Asadi et al. 2005. Balancing performance and reliability in the memory hierarchy. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software 2005 (ISPASS'05)*. IEEE Computer Society, Washington, DC, 269–279. DOI:http://dx.doi.org/10.1109/ISPASS.2005.1430581

G.-H. Asadi et al. 2006. Vulnerability analysis of L2 cache elements to single event upsets. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings (DATE'06)*. European Design and Automation Association, Leuven, Belgium, 1276–1281.

Sanghyeon Baeg, ShiJie Wen, and R. Wong. 2009. SRAM Interleaving distance selection with a soft error failure model. *IEEE Transactions on Nuclear Science* 56, 4 (2009), 2111–2118. DOI:http://dx.doi.org/10.1109/TNS.2009.2015312

R. Baumann. 2005. Soft errors in advanced computer systems. *IEEE Design Test of Computers* 22, 3 (2005), 258–266. DOI:http://dx.doi.org/10.1109/MDT.2005.69

Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri. 2005. Mparm: Exploring the multi-processor soc design space with systemc. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 41, 2 (2005), 169–182.

Binkert and others. 2011. The Gem5 simulator. *SIGARCH Computer Architecture News* 39, 2 (Aug. 2011), 1–7. DOI:http://dx.doi.org/10.1145/2024716.2024718

Doug Burger and Todd M. Austin. 1997. The SimpleScalar tool set, version 2.0. *SIGARCH Computer Architecture News* 25, 3 (June 1997), 13–25. DOI:http://dx.doi.org/10.1145/268806.268810

Yuan Cai, M. T. Schmitz, and others. 2006. Cache size selection for performance, energy and reliability of time-constrained systems. In *Design Automation, 2006. Asia and South Pacific Conference on Design Automation 2006.* 6pp.

DARPA. 2014. Vulture Program. Retrieved from http://www.darpa.mil/Our_Work/TTO/Programs/Vulture.aspx.

Digilent. 2014. Zedboard Data Sheet Overview. Retrieved from http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

A. Dixit and Alan Wood. 2011. The impact of new technology on soft error rates. In *Proceedings of the 2011 IEEE International Reliability Physics Symposium (IRPS)*. 5B.4.1–5B.4.7. DOI:http://dx.doi.org/10.1109/IRPS.2011.5784522

Gaisler. 2014. Leon Processor. (2014). http://www.gaisler.com.

E. Ibe et al. 2010. Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electronic Devices* 57, 7 (2010), 1527–1538. DOI:http://dx.doi.org/10.1109/TED.2010.2047907

JEDEC. 2006. Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices. *JESD89A* (Oct. 2006).

JEDEC. 2007. Test method for beam accelerated soft error rate. *JESD89-3A* (Nov. 2007).

Austin Lesea and others. 2014. Soft error study of ARM SoC at 28 nanometers. In *Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects 2014 (SELSE 10)*.

P. Liden et al. 1994. On latching probability of particle induced transients in combinational networks. In *Digest of Papers on the 24th International Symposium on Fault-Tolerant Computing 1994 (FTCS-24).* 340–349. DOI:http://dx.doi.org/10.1109/FTCS.1994.315626

Shih-Lien Lu et al. 2012. Scaling the memory wall: Designer track. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'12)*. 271–272. DOI:http://dx.doi.org/10.1145/2429384.2429437

J. Maiz et al. 2003. Characterization of multi-bit soft error events in advanced SRAMs. In *Proceedings of the IEEE International Electron Devices Meeting 2003. (IEDM'03 Technical Digest)*. 21.4.1–21.4.4. DOI:http://dx.doi.org/10.1109/IEDM.2003.1269335

Mehrtash Manoochehri et al. 2011. CPPC: Correctable parity protected cache. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*. ACM, New York, 223–234. DOI:http://dx.doi.org/10.1145/2000064.2000091

Shubhendu S. Mukherjee, Joel Emer, Tryggve Fossum, and Steven K. Reinhardt. 2004. Cache scrubbing in microprocessors: Myth or necessity? In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*. IEEE Computer Society, Washington, DC, 37–42.

NASA. 2014. NASA Launches Next Generation PhoneSat. Retrieved from http://www.nasa.gov/content/nasa-launches-next-generation-phonesat-ames-developed-launch-adapter/.

N. Oh, P. P. Shirvani, and E. J. McCluskey. 2002. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability* 51, 1 (2002), 63–75. DOI:http://dx.doi.org/10.1109/24.994913

David A. Patterson and John L. Hennessy. 2013. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA.

H. Quinn and P. Graham. 2005. Terrestrial-based radiation upsets: A cautionary tale. In *Proceedings of the 13th Annual IEEE Annual Symposium on Field-Programmable Custom Computing Machines 2005 (FCCM 2005)*. 193–202. DOI:http://dx.doi.org/10.1109/FCCM.2005.61

M. Rebaudengo, M. Sonza Reorda, and M. Violante. 2003. An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1 (DATE'03)*. IEEE Computer Society, Washington, DC, 10602.

P. Rech et al. 2014. Impact of GPUs parallelism management on safety-critical and HPC applications reliability. In *Dependable Systems and Networks (DSN) 2014*. IEEE.

R. Vemu, S. Gurumurthy, and J. A. Abraham. 2007. ACCE: Automatic correction of control-flow errors. In *Proceedings of the IEEE International Test Conference 2007 (ITC'07)*. 1–10. DOI:http://dx.doi.org/10.1109/TEST.2007.4437639

J. F. Ziegler et al. 1996. IBM experiments in soft fails in computer electronics (1978–1994). *IBM Journal of Research Devices* 40, 1 (Jan. 1996), 3–18. DOI:http://dx.doi.org/10.1147/rd.401.0003

Received September 2014; revised June 2015; accepted June 2015