ECS an Endeavor Towards Providing Similar Cache Reliability Behavior in Different Programs

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

04-06-2021 / 01-11-2021

CITATION

Ahmadilivani, Mohammad Hasan; Moeini Jahromi, Mohammad; E. Salehi, Mostafa; Kargar, Mona (2021): ECS an Endeavor Towards Providing Similar Cache Reliability Behavior in Different Programs. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.14730219.v3>

DOI

[10.36227/techrxiv.14730219.v3](https://doi.org/10.36227/techrxiv.14730219.v3)

ECS an Endeavor Towards Providing Similar Cache Reliability Behavior in Different Programs

Mohammad Hasan Ahmadilivani¹, Mohammad Moeini Jahromi¹, Mostafa E. Salehi¹, Mona Kargar²

¹ School of Electrical and Computer Engineering, Engineering Faculty, University of Tehran, Tehran, Iran

¹{mhasan.ahmadi, moeini.jahromi, mersali}@ut.ac.ir

² Institute of Material and Energy, Tehran, Iran

² mona.kargar@gmail.com

Abstract— The reliability of embedded processors is one of the major concerns in safety-critical applications. Reliability is particularly expressed within the cache memories which are the largest part of new system on chips. Cache memories are the most vulnerable parts of the embedded systems and can affect the reliability drastically especially in deep transistor scaling. Therefore, evaluating the cache vulnerability is crucial in the design of a reliable system especially for safety-critical applications.

It has been shown that using the same cache sizes for different programs leads to incompatible vulnerability patterns in them. According to the literature, most of the related researches, have exploited identical cache sizes for different programs in their reliability evaluations, while the cache reliability strictly depends on the cache size and program behavior. Traditional attempts for finding an appropriate cache size for different programs would need a huge design space exploration.

In this work, we have introduced a criterion for determining the Effective Cache Size (ECS) for embedded processors which considers the inherent programs' reliability and performance properties. According to the results, using the ECS for the representative benchmark applications, the reliability would be increased 43x on average with acceptable performance degradations (21% on average).

Index Terms— Reliability Evaluation, Cache Vulnerability Analysis, Failure in Time, Reliability-Performance Trade-off

I. INTRODUCTION

Nowadays, technology advances lead to the production of complex electronic systems, and applications of embedded systems are increasingly expanded. Nevertheless, technology scaling has increased the vulnerability of new devices against real-world unintended data corruptions. Safety-critical digital systems are supposed to work correctly in real-world conditions, therefore, it is crucial to provide fault-tolerance and high reliability while meeting applications requirements. System reliability is an attribute that expresses the probability of a system working correctly in a given interval of time [1].

Faults and errors influence system reliability because they may corrupt the data or system logics. Soft errors and transient faults are caused by a charged particle (e.g. alpha or neutron) striking transistors. They can induce a tiny amount of voltage to the transistors and flip one or more bits in either logic or memory cells [2]. Regarding the process variation and transistor

scaling, soft errors are prominent concern in digital systems and System on Chips (SoCs).

Memory components that are made up of several transistors in a dense and large area are particularly more exposed to soft errors. Static Random Access Memories (SRAMs) are more susceptible to soft errors than DRAMs and non-volatile memories (i.e. flash memories) [3]. However, SRAMs are mostly used in SoCs because of their lower access time [3]. SRAM-based cache memories are the most susceptible part of the SoCs to soft errors and can affect the reliability drastically [4]. Hence, protecting cache data from soft errors should be considered at design time especially for safety-critical applications such as automotive or aerospace applications.

Protecting caches against soft errors is mostly based on Error Correcting Codes (ECCs) [6-8]. Since ECCs impose energy consumption and area overhead and also degrade performance [5], several researches focused on minimizing the overheads of ECCs in the cache [6-8] and it is still a challenging topic. Nevertheless, to meet performance requirements, implying ECCs in first level cache (cache L1) may lead to great challenges [6]. Hence, it is still an open challenge to develop novel techniques to improve the reliability of unprotected caches as well as protected caches in order to cope with the considerable protection overheads [9]. To this aim, a metric for evaluating the reliability of cache should also be adopted that considers cache reliability along with performance constraints.

Evaluating the cache reliability is a complicated task in the design of digital systems. To model the reliability of the cache, the vulnerability of cache is introduced which is based on the Architectural Correct Execution (ACE) bits and Architectural Vulnerability Factor (AVF) [10]. Cache vulnerability is considered as the summation of all vulnerable intervals. A vulnerable interval is introduced as an interval that a data corruption in that interval, may cause an error that would be propagated through the system. Exploiting vulnerability of cache, we consider the Failure In Time of the cache (FIT_{cache}) as the reliability of cache.

Based on the aforementioned cache reliability model, there are many parameters pertaining to the cache architecture that would affect its reliability [11-13]. In addition, the pattern of program memory accesses influences the vulnerability of cache [14]. It is shown that among the various cache architecture parameters, the cache size is the most effective parameter on the vulnerability of cache [12]. On the other hand, programs vary in memory requirements, and different applications on the

Fig. 1. Cache reliability over identical cache sizes of three programs.

same cache size would have different vulnerability patterns [12].

The reliability of the cache and the effectiveness of any mechanism for improving the cache reliability depend on both cache size and program behavior in accessing its data. According to the literature, most of the related researches have exploited identical cache sizes for different programs in their cache reliability evaluation [12, 13, 15, 23], while the cache reliability strictly depends on the cache size and behavior of the program. Fig. 1 illustrates the cache reliability for three representative programs over identical cache sizes. Regarding Fig. 1, the reliability of each program can be split into three phases over the cache size: low, transient, and high. As shown, the reliability phases of the cache in different programs, highly depend on the cache size for that program. Since the cache reliability phases of the programs are different in each size, it is not fair to compare the reliability or the effects of reliability improvement techniques in the same sizes for all programs. Rather it is better to compare them in the same reliability phases.

Some researchers have tried to propose different optimum cache sizes for different programs [16, 17]. Their methods are based on exploring the design space of cache configurations while running predetermined programs for finding out the proper cache configuration for each program. Nevertheless, these researches don't present a general criterion for determining the proper cache size for different programs. In addition, these methods need exhaustive explorations on different cache configurations for each program to find the suitable configuration and hence, suffer from time-consuming exhaustive design space explorations.

In this work, we define the "Effective Cache Size" (ECS) and its coefficients in which the performance and reliability phases of all programs are almost similar. As a motivating example, considering the same programs that are used in Fig. 1, the pattern of cache reliability based on coefficients of ECS is shown in Fig. 2. It is observed that the phases of cache reliability are almost similar in different coefficients of the ECS, although the sizes are not identical. Therefore, based on Fig. 2, reliability and its improvement techniques for different programs could be fairly compared in the same coefficients of the ECS.

Therefore, by knowing the ECS of each program, a system with a reconfigurable cache may modify its cache size to the proper size by turning on and off its memory banks before

Fig. 2. Cache reliability over effective cache sizes of three programs.

starting to run a program. In this case, not only the performance and reliability could be improved, but also the power would be optimized. As a matter of fact, The proposed ECS is not an alternative to ECC methods of reliability improvement. However, it is an intuitive criterion which considers the effect of programs memory footprint on the reliability as well as the structure of the cache and provides a fair condition for evaluating the reliability improvement methods.

To find out a criterion for determining the ECS for various programs, we have investigated the cache access patterns of several programs considering cache reliability and performance tradeoffs. As a summary, our contributions in this work are:

- 1- distinguishing the inconsistency of reliability phases of different programs in an identical cache size by adopting an analytical cache reliability model,
- 2- finding a common relationship between different programs' address space and their access patterns in order to perceive a general criterion for determining proper cache size for different programs,
- 3- proposing an intuitive criterion as Effective Cache Size (ECS) such that reliability and performance behavior of different programs are almost similar in the same coefficients of the ECS,
- 4- presenting a fast method to obtain the ECS for different programs and hence, avoiding time-consuming design space explorations.

The remainder of this paper is as follows: related works are discussed in section II, motivation and observation are mentioned in section III followed by the simulation setup in section IV, and the experimental results are discussed in section V. At last, section VI represents the conclusion of this work.

II. RELATED WORKS

A. Cache Reliability Model

To estimate the reliability of cache, an integrated model of cache reliability is needed. To this aim, we adopt an analytical model for cache reliability. The reliability of a structure in a digital system is modeled based on the (1) [2]:

$$FIT_{structure} = FIT_{intrinsic} \times N \times AVF_{structure} \quad (1)$$

where $FIT_{intrinsic}$ as the number of Failure In Time (FIT) is a technology-dependent constant for the device, N represents the structure size regarding its number of bits, and $AVF_{structure}$ is the Architectural Vulnerability Factor of the structure. AVF is introduced as the probability of error propagation at the

architecture level. In the concept of the cache, Cache Vulnerability Factor (*CVF*) represents the AVF which is introduced in [10] and expresses the probability of error propagation from the cache to other components of the system (e.g. CPU and other levels of memory hierarchy). *CVF* is formulated in (2):

$$CVF = \frac{\text{cache vulnerability}}{\text{execution time} \times \text{cache size}} \quad (2)$$

Cache vulnerability is defined as the total cycles of vulnerable intervals between the accesses to a cache that end with a read or evict as shown in Fig. 3. The faults that have occurred in a vulnerable cache interval, may be propagated to other components of the system as an error [18]. Hence, based on (1) and (2) the reliability model of the cache is shown in (3):

Fig. 3. Vulnerable periods of a data in cache.

$$FIT_{\text{Cache}} = FIT_{\text{intrinsic}} \times \text{cache size} \times CVF \quad (3)$$

By replacing the *CVF* of (3) with (2), (4) can also represent the reliability of cache:

$$FIT_{\text{Cache}} = \frac{FIT_{\text{intrinsic}} \times \text{vulnerability}}{\text{execution time}} \quad (4)$$

Based on (4), the reliability of the cache has a direct relationship with the cache vulnerability. Nevertheless, in the terminology of this paper, the cache reliability refers to FIT_{Cache} . We have proposed a vulnerability estimation method in our previous work [19] which is more accurate compared to [20] due to considering dirty bit for each word of data in our work.

B. Cache Reliability Exploration

As mentioned by [11], Data cache (Dcache) is more vulnerable than Instruction Cache (Icache) because it contains processed and dirty data that are vulnerable until they are written back to higher cache levels. On the other hand, the instruction cache has a duplicated data in lower memories, therefore, protecting Dcache is more crucial. That is why we consider Dcache reliability concerns in this work.

Some researches explore the most effective parameters on the cache vulnerability including cache size, write-hit policy, replacement strategies, associativity, and so on [12-15, 18]. It has been observed that cache size is the most effective parameter on the cache vulnerability [12]. Cai et. al. [13] have performed some simulations to find the best cache size regarding the tradeoff between performance, energy, and reliability. Based on their simulation results, the cache size has a remarkable effect on the reliability and performance of the cache. However, the results of [13] are based on identical cache sizes for different benchmarks.

In another research, Kooli et. al. [12] have studied the effect of different structures and policies on the cache vulnerability

for different programs. The results are summarized as follows: 1) Cache size is the most effective parameter on the cache vulnerability. 2) When a program utilizes a larger cache for better performance, the reliability decreases, otherwise on smaller caches we have lower performance and better reliability. 3) The trend of performance over different cache sizes is not identical among different programs. Therefore, the vulnerability behavior of programs is also various in different cache sizes. Hence, considering identical cache size for studying the reliability of different programs is not fair.

Huang et. al. [16] try to use different cache configurations for different programs to manage the cache vulnerability considering energy optimizations. They find an optimum cache size with performance and energy constraints. Furthermore, their method is based on the exploration of each program and there is no general criterion to limit the time-consuming design space explorations.

It is shown that the evaluation of ECC-based approaches in memory arrays are performed at design time by executing benchmarks with fixed and predetermined cache sizes [23]. Therefore, the effect of cache size on the cache vulnerability of programs is not considered, while the cache size is the most effective parameter on the cache vulnerability.

To have a fair comparison, there should be a general criterion for determining the proper cache size for different programs. To the best of our knowledge, except some works that used exhaustive design space explorations, there is no general criterion for approximating the reliability phases of Dcache for different programs. To this end, in this work, we have investigated the memory footprint and cache access patterns for several programs and provided a general criterion to determine the effective cache size.

III. MOTIVATION AND OBSERVATIONS

A. Reliability-Performance Tradeoff in Identical Cache Sizes for Different Programs

Fig. 4 shows the Dcache reliability (based on FIT_{Cache} in (4)) for 10 representative benchmark programs with different 1-way cache sizes. In this case, some fixed sizes are set for Dcache and the value of FIT_{Cache} at each point is normalized to the FIT_{Cache} of the basic cache (the minimum cache size that gains almost the lowest execution time of the program). As shown in Fig. 4, in small cache sizes (e.g. 1 to 4 KB), FIT_{Cache} of programs are close to each other. However, based on Fig. 5, which shows the execution time of programs normalized to the lowest execution time of each, the performance of programs in those cache sizes (1 to 4 KB) are in a wide range and most of them will be executed in at least 2x more cycles than their lowest execution time.

On the other hand, at larger sizes where the execution times of programs are low (in 512 KB or larger the normalized execution time is near 1 in Fig. 5), the normalized FIT_{Cache} has a wide range in different programs (in Fig. 4). For instance, the normalized FIT_{Cache} of qsort at 32k-1w Dcache is 2% while it is 97% for bitcount. Therefore, it is observed that the normalized

Fig. 4. Trend of normalized FIT_{cache} of programs on different sizes of direct-mapped Dcache.

FIT_{cache} and execution time of programs vary considerably in different cache sizes.

Based on this observation, if the cache size is large enough, the reliability is low because data will remain in the cache for a longer time and if any fault happens, it may be propagated with higher probability. However, if the cache size is too small, the performance would be degraded remarkably. By finding a suitable cache size which can accommodate enough data as is needed, then both performance and reliability tradeoffs would be met. As a matter of fact, it can be claimed that if the patterns of FIT_{cache} of different programs are close to each other at a golden size in which the performance is also good, the criterion for finding the aforementioned cache size is disclosed.

B. Effect of Input Data on the Cache Reliability

Input data may affect both the control and data flow of programs. Therefore, their impact on the reliability of cache should be investigated. As an example, we run bubble sort on 50 elements in Gem5 with 10 random input vectors and computed FIT_{cache} based on (4) as will be mentioned in section IV.A. As a result, we found that different input data do not have a considerable impact (less than 0.1%) on the FIT_{cache} over different input data to this program. Hence, we can conclude that input data do not influence the cache reliability based on the proposed model. So that we will use the default input data of the representative benchmarks, for the sake of feasibility and repeatability of this work.

C. Access Patterns of Representative Programs

Based on the previous observation, the cache reliability varies a lot for different programs. which is caused by their different cache access characteristics. We have analyzed the data addresses of programs and as an example, Fig. 6 shows the histogram of accessed addresses in the bubble sort program on an array of 50 elements. The right side shows the accumulative top accesses to the more frequent and hot addresses of the program. The left side shows the relevant accumulated ratio of the number of more frequent addresses to the total number of accessed addresses all over the program execution. For example, 0.1%, 0.25%, 0.70%, and 1.1% of program addresses have contributed to 20%, 40%, 65%, and 80% of the total number of accesses to the cache, respectively.

Based on Fig. 6 and a similar analysis on different programs, it can be concluded that a little number of addresses contribute to the most top cache accesses of the program. Therefore, a

Fig. 5. Trend of normalized execution time of programs on different sizes of direct-mapped Dcache.

small number of addresses of a program constitute the majority of accesses. The mentioned observation has been investigated for more programs and the results are exported from profiling of representative programs in the Gem5 simulator. Based on this analysis, we observed that in most of the programs, accumulated top accesses to the addresses are saturated at 95%. It means that 95% of program accesses are made by a few numbers of the whole addresses of the program. Hence, if these data reside in the cache, the performance would be acceptable due to the high frequency of accesses to these data.

Therefore, the information of the range between minimum to maximum data addresses (*range*) and the number of unique accessed data addresses (*count*) of programs are summarized in Table 1 as well as number of addresses that contribute to 95% of the total accesses to cache (*top95*). As shown in Table 1, in most of the programs, 95% of accesses are to a small portion of the address space. Considering the accessed addresses of programs leads to interesting observations:

- 1- In each program, a very small portion of the address space is accessed most of the time,
- 2- Address space or number of unique addresses are program-specific representative attributes for any program and can be used for selecting the cache size for different programs.

Fig. 6. Access pattern of bubble sort.

Based on the previous observations, if the cache size is large, the vulnerability is very high. On the other hand, a small cache size reduces the system performance. Hence, to find the effective cache size (ECS), if we introduce a cache size that can accommodate as many data of the program as enough, both performance and reliability requirements would be met.

IV. SIMULATION SETUP AND PROPOSED METHOD

A. Simulation Environment

In this work, we have used Gem5 [21] simulator to run the programs with different configurations for cache in System Emulator (SE) mode. The processor is X86 with 1 GHz frequency. The processor has separated instruction and data cache. The Icache is considered constant (16 KB with random replacement policy), however, different sizes of Dcache ranging from 512B to 2MB are explored. The associativity of Dcache is direct-map and 8-way set associative to evaluate and validate the results in the application domain. The replacement policy is LRU and the cache block size is 64 words. To meet the generality of the idea, several programs from MiBench [22] benchmark are examined including: basicmath, bitcount, crc, dijkstra, fft, patricia, qsort, string-search, sha, and susan.

TABLE 1. STATISTICS OF THE RANGE, COUNT, AND TOP95 OF REPRESENTATIVE PROGRAMS.

Program	Range	Count	Top95
basicmath	443.5 KB	68.8 K	3600
bitcount	825 KB	5.3 K	200
crc	4.3 MB	108 K	1400
dijkstra	941 KB	65.6 K	3700
fft	986.7 KB	76.1 K	3400
patricia	2 MB	878.6 K	7500
qsort	8.4 MB	1.4 M	30000
strsearch	0.99 MB	61.6 K	1900
sha	1.4 MB	104.5 K	900
susan	1.4 MB	35.1 K	450

B. Cache Vulnerability Estimation

To estimate the cache vulnerability introduced in (4), we have exploited algorithm which depicted in Fig. 7. In the vulnerability estimation model, we have assumed that each word has its own dirty bit. Based on this algorithm which is an implementation of Fig. 3, all vulnerable periods of each word in the cache would be considered separately and exactly. This algorithm is implemented in python and uses profiling of Gem5's output logs on programs' accesses to the cache.

Fig. 7. Flow chart of cache vulnerability estimation algorithm

C. Baselines

We have presented FIT_{cache} and execution time in normalized values in this work. A normalized value for FIT_{cache} (or execution time) in a specific size is obtained by dividing the value of FIT_{cache} (or execution time) at the target cache size into the value of FIT_{cache} (or execution time) in the basic Dcache size of the program.

Basic Dcache size is the minimum size of the cache for a program which almost leads to the best performance, so that increasing the cache size to the values greater than the basic cache size does not further improve the performance. Basic Dcache size of each program is shown in Table 2.

D. Proposed Method

Based on the memory address space and program cache accesses that are shown in Table 1, three criteria are considered for determining the cache size: 1) Range: the range of addresses which a program will access, 2) Count: the number of all unique addresses which are accessed by the program, 3) Top95: accumulative number of addresses which have contributed to 95% of cache accesses of a program.

According to the mentioned criteria, three different methods of determining the cache size are presented. Table 2 shows the Dcache sizes for any program in each criterion. Cache sizes are chosen based on Table 1 in a way that they can fit all data regarding the corresponding criteria. We used each of the above criteria to determine the cache size and considered some coefficients of them (e.g. 1/8x, 1x, 4x, etc.). Table 2 shows the 1x value of Dcache size for each program in each criterion.

TABLE 2. DCACHE SIZE DETERMINATION BASED ON DIFFERENT CRITERIA.

program	Dcache size 1x			Basic Dcache size
	Range	Count	Top95	
basicmath	512 KB	64 K	4 K	256 KB
bitcount	1 MB	4 K	256	128 KB
crc	4 MB	128 K	2 K	512 KB
dijkstra	1 MB	64 K	4 K	256 KB
fft	1 MB	64 K	4 K	512 KB
patricia	2 MB	1 M	8 K	2 MB
qsort	8 MB	1 M	32 K	2 MB
strsearch	1 MB	64 K	2 K	512 KB
sha	2 MB	128 K	1 K	512 KB
susan	2 MB	32 K	512	1 MB

To select one of these criteria as the ECS, we consider normalized FIT_{cache} and execution time of programs. The set of points which pass both reliability and performance constraints will be selected as candidates for ECS. In the final step, the criterion that has the least variance in normalized FIT_{cache} in different programs will be selected as the ECS.

V. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we present the results of the proposed criteria for finding the ECS. Normalized FIT_{cache} and execution time (as mentioned in IV-C) of programs based on different coefficients of the *Range* criterion are presented in Fig. 8 (1x values are shown in the column of *Range* in Table 2). As it can be observed in Fig. 8-a, the phases of the cache reliability of different

Fig. 8. Normalized FIT_{Dcache} and execution time of programs considering *Range* coefficients as cache size for direct-map cache, a) FIT_{cache} , b) Execution Time.

programs are neither similar nor close. At the point where the normalized FIT_{cache} values are low and close in Fig. 8-a (1/256x where all FIT_{cache} of programs are less than 0.4), execution times of programs are 1 to 2.23 times more than the minimum execution time of the corresponding program. On the other hand, at 1/8x in Fig. 8-b where the normalized execution time of almost all programs are good, the normalized FIT_{cache} of programs varies from 0.6 to 1 which are close to the worst case reliability of them. Hence, with *Range* coefficients, we cannot find a proper size for cache that satisfies both reliability and performance constraints. We have concluded to the same analysis using 8-way caches, to save space we have not presented the results of 8-way caches here.

Results of normalized FIT_{cache} and execution time of programs based on *Count* and *Top95* criteria are presented in Fig. 9 (a-h) for both direct-map and 8-way configurations. The coefficients of *Count* (1/64x to 8x) and *Top95* (1/2x to 256x) are specified regarding the values of the sizes in the corresponding column in Table 2. As shown in Fig. 9-a, considering direct-map cache and *Count* coefficients as the effective cache size, the reliability phases seem to be close in different programs. If we consider the performance results in Fig. 9-c, two points of coefficients can be selected based on this criterion: 1/16x and 1/8x, where all normalized FIT_{cache} results are less than 0.4 and normalized execution times are less than 2. At the mentioned points, the values of cache reliability of different programs are not only close to each other but also are improved (16.4x and 10x reliability improvement in average, on 1/16x-1w and 1/8x-1w of *Count* configurations respectively), trading endurable performance degradation (45% to 27% performance degradation in average, on 1/16x-1w and 1/8x-1w configurations of *Count* respectively) with respect to the basic cache of each program.

Fig. 9-b and Fig. 9-d illustrate the reliability and performance of programs considering direct-map cache with *Top95* coefficients as effective cache size. As shown in Fig 8-b, the reliability phases of programs are closer than *Count* and the normalized value of FIT_{cache} is in a low and limited range. Regarding the normalized execution time of programs in Fig. 9-d, the best cache sizes can be 2x and 4x (where all results of FIT_{cache} are less than 0.4 and normalized execution time is less

than 2). At the mentioned points, the values of cache reliability of different programs are improved (28.5x to 10.5x reliability improvement on 2x-1w and 4x-1w of *Top95* configurations respectively), trading endurable performance degradation (36% to 21% performance degradation on 2x-1w and 4x-1w of *Top95* configurations respectively) with respect to the basic cache of each program.

Fig. 9-e and Fig. 9-g demonstrate the normalized FIT_{cache} and execution time of programs considering 8-way caches and *Count* coefficients as the effective cache size criterion. If we consider both reliability and performance constraints, two points of these figures can be proposed: 1/16x and 1/8x where FIT_{cache} is less than 0.4 and normalized execution time is less than 1.5. At the mentioned points, the values of cache reliability of different programs are improved (21.3x and 7x reliability improvement on 1/16x-8w and 1/8x-8w of *Count* configurations respectively), trading endurable performance degradation (13% and 8% performance degradation on 1/16x-8w and 1/8x-8w of *Count* configurations respectively) with respect to the basic cache of each program.

The results of normalized FIT_{cache} and execution time for *Top95* criterion in 8-way configuration are also shown in Fig. 9-f and Fig. 9-h respectively. Considering the previous method on finding the suitable coefficients, the points for 1x and 2x coefficients could be chosen as the effective cache size. At the mentioned points, the values of cache reliability of different programs are improved (43.2x and 23.2x reliability improvement on 1x-8w and 2x-8w of *Top95* configurations respectively), trading endurable performance degradation (21.3% to 11.4% performance degradation on 1x-8w and 2x-8w of *Top95* configurations respectively) with respect to the basic cache of each program.

It is worth mentioning that the value of 0.4 for normalized FIT_{cache} which was observed within the figures, is merely obtained by the experiments and it is not a constraint to reach to the ECS. But still, it can be considered analytically as a good value for the normalized FIT_{cache} because it is expressing that the cache is more reliable at least 2.5 times than the basic cache. In the same way, the reported values on performance degradation (based on normalized execution time) are described analytically. Nevertheless, constraints of reliability and

Fig. 9. Normalized FIT_{cache} and execution time of programs considering coefficients of *Count* and *Top95* as cache size: a) FIT_{cache} considering coefficients of *Count* with 1-way cache, b) FIT_{cache} considering coefficients of *Top95* with 1-way cache, c) Execution time considering coefficients of *Count* with 1-way cache, d) Execution time considering coefficients of *Top95* with 1-way cache, e) FIT_{cache} considering coefficients of *Count* with 8-way cache, f) FIT_{cache} considering coefficients of *Top95* with 1-way cache, g) Execution time considering coefficients of *Count* with 8-way, h) Execution time considering coefficients of *Top95* with 8-way cache.

performance are determined by the application. Table 3 shows a summary of the results of the selected points correspond to the reliability improvement and performance degradation in different cache configurations and size determination criteria (Count and Top95).

To find out which point in the above criteria (*Count* and *Top95*) is the best candidate for ECS, the variance of FIT_{cache} in different programs are compared and the ECS which has the least variance of FIT_{cache} will be chosen. Fig. 10 shows the variance of FIT_{cache} of selected points in Fig. 9-a and Fig. 9-b. Also Fig. 11 illustrates the variance of selected points in Fig. 9-e and Fig. 9-f. As shown, *Top95* criteria has the best result in both configurations. In the direct-map cache, size of 2x of *Top95* and in 8-way configuration, size of 1x of *Top95* are the best candidates as the ECS for programs.

TABLE 3. RESULTS OF RELIABILITY IMPROVEMENT AND PERFORMANCE DEGRADATION FOR SELECTED POINTS OF COUNT AND TOP95 CRITERIA IN DIFFERENT CACHE CONFIGURATIONS.

Cache Configuration	Average Execution Time Degradation	Average FIT Improvement
1/16x-1w Count	45%	16.4x
1/8x-1w Count	27%	10x
2x-1w Top95	36%	28.5x
4x-1w Top95	22%	10.5x
1/16x-8w Count	13%	21.3x
1/8x-8w Count	8%	7x
1x-8w Top95	21%	43x
2x-8w Top95	11.4%	23.2x

Fig. 10. FIT_{cache} variance for the selected points considering *Count* and *Top95* for direct-map cache.

Fig. 11. FIT_{cache} variance for selected points considering *Count* and *Top95* for 8-way cache.

Table 4 shows the summary of results with respect to the ECS obtained by the aforementioned results. As a summary, if we determine the cache size based on *Top95* criterion, in the direct-map configuration we will gain 28.5 times more reliability than the basic cache of each program with the cost of 36% longer execution time in average. Meanwhile, if the configuration of Dcache is 8-way set associative, the results are

TABLE 4. RELIABILITY AND PERFORMANCE RESULTS OF ECS

ESC	2x-1w Top95	1x-8w Top95
Min normalized FIT	0.019	0.007
Max normalized FIT	0.066	0.055
Average FIT	0.035	0.023
Average reliability improvement	28.5x	43x
Min normalized performance degradation	1.01	1.01
Max normalized performance degradation	1.86	1.55
Average performance degradation	36%	21%

remarkably better. The reliability would gain 43 times better than the basic cache and the execution time of programs would take 21% longer time.

VI. CONCLUSION

One of the most vulnerable components of embedded systems is cache, which plays a major role in determining the vulnerability of the entire system. Cache parameters affect its vulnerability, and as mentioned, its size is the most effective parameter in determining its vulnerability. On the other hand, cache size plays an important role in system performance. Therefore, determining the cache size is a trade-off between vulnerability and system's performance. The size of cache has different effects on the vulnerability of different programs, depending on the amount of memory required by the program and its access pattern. Accordingly, choosing the same size for different programs is not an appropriate method. In case, a processor with a reconfigurable cache, can reconfigure the cache before running a program by given information about the suitable cache size.

We introduce the Effective Cache Size so that the reliability behavior of different programs is similar in the same ECS coefficients. To determine this criterion, we have examined the address space and access pattern of different programs. For this purpose, three criteria for selecting the cache size are introduced and examined: *Range*, *Count* and *Top95*. The *Range* criterion is not a suitable criterion to assimilate the reliability behavior of different programs. The *Top95* criterion achieves the best results both in approximating the reliability status of different programs and in reducing performance.

Based on results, ECS is 1x and 2x of *Top95* size of the program in the direct-map configuration and 8-way set associative configurations, respectively. In the direct-map

configuration, the reliability is 28.5 times better than the basic cache but the programs would be executed 36% longer in average. In the 8-way set associative configuration, the cache is 43 times more reliable than the basic cache with the cost of 21% longer execution time in average. It has also been shown that determining the cache size based on the ECS will provide similar conditions of reliability for different programs in the target cache.

REFERENCES

- [1] B. W. Johnson, "Fault-Tolerant Microprocessor-Based Systems," *IEEE Micro*, vol. 4, no. 6, pp. 6-21, Dec. 1984
- [2] S. Mukherjee, *Architecture Design for Soft Errors*, Elsevier, 2008
- [3] S. Mittal, J. vetter, "Reliability Tradeoffs in Design of Volatile and Nonvolatile Caches," *Journal of Circuits, Systems and Computers*, vol. 25, no. 11, pp. 1-8, Nov. 2016
- [4] C. Slayman, "Soft error trends and mitigation techniques in memory devices," in *Proceedings of Annual Reliability and Maintainability Symposium*, Lake Buena Vista, FL, USA, 24-27 Jan. 2011
- [5] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, J. C. Hoe, "Multi-bit Error Tolerant Caches using Two-Dimensional Error Coding," in *proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, Chicago, IL, USA, 1-5 Dec. 2007
- [6] J. Hong, S. Kim, "Smart ECC Allocation Cache Utilizing Cache Data Space," *IEEE Transaction on Computers*, vol. 66, issue 2, pp. 368-374, Feb. 2017
- [7] I. Alam, C. Schoeny, L. Dolecek and P. Gupta, "Parity++: Lightweight Error Correction for Last Level Caches," in *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Luxembourg, 25-28 Jun. 2018
- [8] P. Benedicte, C. Hernandez, J. Abella and F. J. Cazorla, "LAEC: Look-Ahead Error Correction Codes in Embedded Processors L1 Data Cache," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019
- [9] F. Kriebel, S. Rehman, A. Subramaniyan, S. J. B Ahandagbe, M. Shafique, J. Henkel, "Reliability-Aware Adaptations for Shared Last-Level Caches in Multi-Cores," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 4, article 67, August 2016
- [10] S. Mukhejee, J. Emer, S.K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in *proceedings of 11th International Symposium on High-Performance Architecture*, USA, 12-16 Feb. 2005
- [11] Y. Ko, K. Lee, "Multi-Level Cache Vulnerability Estimation: The First Step to Protect Memory," in *proceedings of 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Hungary, 9-12 Oct. 2016
- [12] M. Kooli, G. Di Natale, A. Bosio, "Memory-Aware Design Space Exploration for Reliability Evaluation in Computing Systems," *Journal of Electronic Testing* 35, pp. 145-162, 2019
- [13] Y. Cai, M. T. Schmitz, A. Ejali, B. M. Al-Hashimi and S. M. Reddy, "Cache size selection for performance, energy and reliability of time-constrained systems," in *Proceedings of Asia and South Pacific Conference on Design Automation*, Yokohama, Japan, 24-27 Jan. 2006
- [14] A. Shrivastava, J. Lee, and R. Jeyapaul, "Cache vulnerability equations for protecting data in embedded processor caches from soft errors," in *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems (LCTES 2010)*, 13-15 Apr. 2010
- [15] J. Yan, W. Zhang, "Evaluating instruction cache vulnerability to transient errors," in *Proceedings of the 2006 workshop on MEMory performance: DEALing with Applications, systems and architectures (MEDEA '06)*, New York, USA, Sep. 2006
- [16] Y. Huang, P. Mishra, "Reliability and energy-aware cache reconfiguration for embedded systems," in *proceedings of 17th International Symposium on Quality Electronic Design (ISQED)*, USA, 15-16 Mar. 2016
- [17] Y. Huang, P. Mishra, "Vulnerability-Aware Energy Optimization for Reconfigurable Caches in Multitasking Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 809-821, May 2019
- [18] W. Zhang, "Computing cache vulnerability to transient errors and its implication," in *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, USA, 3-5 October 2005
- [19] M. H. Ahmadilivani, M. E. Salehi, M. Kargar, "Effect of Cache Run-Time Parameters on the Reliability of Embedded Systems", *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, Tehran, Iran, 10-11 Jun. 2020
- [20] Y. Ko, R. Jeyapaul, Y. Kim, K. Lee, A. Shrivastava, "Protecting Caches from Soft-Errors: A Microarchitect's Perspective," *ACM Transaction on Embedded Computing Systems*, vol. 16, no. 4, article 93, May 2017
- [21] Gem5 simulator (2019). Gem5 home page [online]. Availabel: <https://www.gem5.org/>
- [22] Github (2019). MediaBench source code [online]. Availabel: <https://github.com/t-crest/patmos-benchmarks/tree/master/Mediabench>
- [23] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *International Symposium on Microarchitecture*, 2003