Microelectronics Journal 90 (2019) 123–132

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

CLEAR: Cache Lines Error Accumulation Reduction by exploiting invisible accesses

Hamed Farbeh [a,*], Amir Mahdi Hosseini Monazzah [b]

[a] Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

ARTICLE INFO

Keywords:
Error correction
Multiple-Bit Upset (MBU)
Single-Event Upset (SEU)
Soft error accumulation
Temporal MBU

ABSTRACT

SRAM caches are the most vulnerable processor component to radiation-induced soft errors. Error-Correcting Codes (ECCs) are the conventional scheme to protect caches against soft errors. These errors in the shape of Single-Event Upset (SEU), in which single bit or multiple adjacent bits are affected, can be correctable by ECCs. However, conventional ECCs are unable to correct temporal MBUs (Multiple-Bit Upset), in which two or more SEUs are accumulated in a data word over time. This paper proposes *CLEAR* (Cache Lines Error Accumulation Reduction) scheme to reduce the occurrence probability of temporal MBUs. By exploiting inherently available externally invisible accesses to all cache ways in a read request, CLEAR conducts more frequent ECC checking for each data word. Therefore, the ECC checking intervals are shortened, which leads to a significant reduction in occurrence probability of temporal MBUs. The evaluations show that CLEAR increases the *Mean Time To Failure* (MTTF) of the cache by more than 4× compared to the conventional cache architecture with negligible overheads.

1. Introduction

Soft errors are the main cause of computer systems failure [1]. Radiation-induced particle strikes have long been known as the main source of soft errors [2,3]. These particles are dominated by high-energy neutrons from cosmic rays and alpha particles omitted from impurities in chip packaging materials [4–9]. Soft errors can also occur by low energy (thermal) neutrons [7]. Occupying a large fraction of total processor chip area [10,11], SRAM caches are the most susceptible components to soft errors [3,8,9,12–14]. The susceptibility of caches increases with continuous technology scaling [3,4,15–19]. Moreover, reduction in voltage swing, to decrease the power consumption in caches, and increase in temperature further accelerate the soft error rate [9,17,20–23].

The strike of a particle to cache can cause an unintentional change in data value, known as *Single-Event Upset* (SEU) [24,25]. A SEU results in flipping a single memory cell [26], causing *Single-Bit Upset* (SBU) [4,27], or several adjacent memory cells [6,8,9,18,22,27,28], causing spatial *Multiple-Bit Upset* (spatial MBU) [9]. A *temporal MBU* occurs when SEUs are accumulated in a data word over time [4,9,18]. Detection and correction of SBUs, spatial MBUs, and temporal MBUs are a necessity for reliable cache operation [9,17,18,29].

Error Correcting Codes (ECCs) are commonly used to detect and correct soft errors [30–33]. Parity check and Single Error Correction-Double Error Detection (SEC-DED) codes are the most conventional ECCs in caches [3,4,6,30,34–36]. Parity check and SEC-DED are capable of detecting SBUs and correcting SBUs, respectively. To overcome spatial MBUs, three approaches are available: a) transforming a spatial MBU into multiple SBUs by memory bits interleaving [4,6,18,28,37], in which physically adjacent data bits belong to different data words; b) ECC interleaving, in which multiple ECCs that protect a code word interleave their data bits [14,38]; and c) using ECCs customized for correcting adjacent bit errors [28], e.g., SEC-DED-Double Adjacent Error Correction (SEC-DED-DAEC) and SEC-DAEC [33,39]. None of these three approaches is capable of detecting or correcting temporal MBUs [18,40].

To reduce the occurrence probability of temporal MBUs, ECCs can be combined with memory scrubbing scheme [4,18,40]. Scrubbing scheme periodically reads the whole memory contents for ECC checking to prevent the accumulation of SEUs. The occurrence probability

* Corresponding author.
E-mail addresses: farbeh@aut.ac.ir (H. Farbeh), monazzah@ipm.ir (A.M.H. Monazzah).

https://doi.org/10.1016/j.mejo.2019.05.020
Received 27 August 2018; Received in revised form 18 February 2019; Accepted 28 May 2019
Available online 14 June 2019

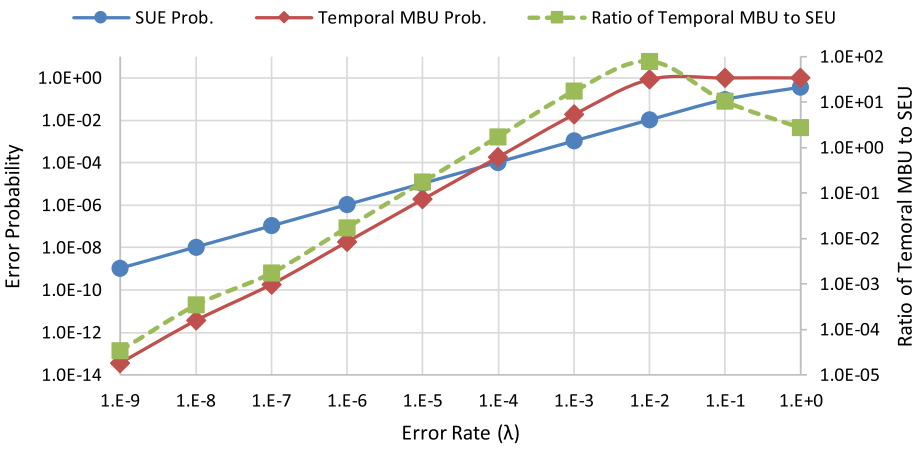


Fig. 1. Occurrence probability of SEU and MBU for a wide range of error rate per memory cell.

of temporal MBUs is directly affected by the interval between two consecutive accesses to a memory block [5] and it can be practically prevented by scrubbing the memory in short enough intervals [40,41]. Scrubbing is widely used in off-chip memories [18,40]. However, this scheme significantly increases the energy consumption and memory traffic when employed in high-speed frequently-accessed on-chip caches [4,14,41].

This paper formulates the occurrence probability of temporal MBUs in ECC-protected caches based on the frequency and interval of accesses to data words. Then, a low-cost scheme is proposed to reduce the occurrence probability of temporal MBUs by conducting more frequent ECC checking for data words. This scheme, so-called *CLEAR (Cache Lines Error Accumulation Reduction)*, exploits an inherent redundancy in the cache for minimizing the access time to increase the frequency of ECC checking. Typically, for a read access to the cache, data words in all cache ways are read out in parallel with accessing and comparing the tags. Then, the output of tag comparison operation selects the requested data word in the target way and the other words are discarded.

Accessing all cache ways beforehand instead of accessing only the requested way overlaps the latency of data access with the latency of tag operations. CLEAR takes advantage of these externally invisible accesses to reduce the probability of SEUs accumulation. The key idea behind CLEAR is to conduct ECC checking for all accessed words instead of checking only the requested word. To this end, CLEAR makes a minor modification in cache read path. This modification significantly reduces the occurrence probability of temporal MBUs in cache words with negligible overhead.

CLEAR is evaluated using the gem5 full-system simulator [42] running the workloads from SPEC CPU2006 benchmark suite [43]. As compared with the conventional cache architecture, CLEAR increases the Mean Time To Failure (MTTF) of the cache by 452%. This improvement is achieved by no performance reduction, less than 2% area overhead, and about 1.5% increase in energy consumption.

The rest of this paper is organized as follows. Section 2 formulates the reliability of the cache under temporal MBU occurrences. The proposed CLEAR scheme is presented in Section 3. The simulation setup and results are given in Section 4. Finally, section 5 concludes the paper.

2. Problem formulation

Cache memories are conventionally capable of correcting single-event errors, using ECCs. ECC check bits are generated for a data word in a write request and these check bits are checked in a read request. We define the failure of the cache as inability of ECC to correct an error. In other words, the reliability of the cache is violated when an erroneous data word is read but is uncorrectable. It is assumed that ECC is capable of correcting SEUs (SBUs and spatial MBUs), but unable to correct temporal MBUs.

To have a comparison between occurrence probability of SEU and MBU, Fig. 1 illustrates the occurrence probability of SEU and temporal MBU per unit of time in a 32-Kbyte cache for a wide range of error rates (λ). As shown, for very low error rate, the MBU probability is by near five orders of magnitude lower than SEU probability. By increasing the error rate, the gap between SEU and MBU probabilities is reduced and around error rate of 10^{-4}, they area almost the same. This means that for the given time interval, the existence probability of cache words affected only once by a particle strike is almost the same as cache words affected by more than single particle strike. For error rates higher than 10^{-4}, the MBU probability is higher than the SEU probability and the gap between the two increase toward the error rate of 10^{-2}. After that, MBU probability is saturated and SEU probability increases to the saturation point. This observation indicates that MBU probability cannot be ignored even for very low error rate of 10^{-9}.

In the following, we calculate the reliability of the cache in the presence of SEUs. Assume that a data word has been written into the cache at time t_1 and then has been read at time t_2. This read access is reliable if the word either is error-free or a SEU has occurred. Considering the exponential distribution for soft errors with the rate of λ, the probability of a word to be error-free is according to (1):

$$p(n = 0) = e^{-\lambda t} \tag{1}$$

Where n is the number of SEUs, t is a time interval equal to $t_2 - t_1$, and λ is the SEU rate. The probability that a single error (one SEU) occurs in the word in interval t follows a *Poisson distribution* and is calculated according to (2):

$$p(n = 1) = \lambda t e^{-\lambda t} \tag{2}$$

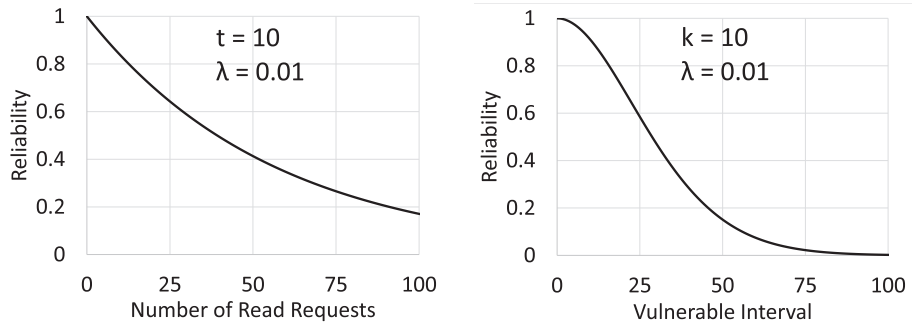
The probability that the word is either error-free or correctable on a read access is the summation of (1) and (2). Therefore, the reliability of the word in interval t (or the reliability of the cache for this access) is calculated by (3):

$$R_{word} = p(n \le 1) = e^{-\lambda t} + \lambda t e^{-\lambda t} = (1 + \lambda t)e^{-\lambda t} \tag{3}$$

Interval t is the time between the last write/read request performed correctly and the current read request for a data word. In this interval, a data word is vulnerable to soft errors. For a given vulnerable interval t, the reliability of the cache for all read requests to data words whose vulnerable interval duration equals to t is calculated by (4):

$$R_{k_t}(t) = ((1 + \lambda t)e^{-\lambda t})^{k_t} = (1 + \lambda t)^{k_t} e^{-\lambda t k_t} \tag{4}$$

Where k_t is the total number of read requests to data words that their vulnerable interval is t. This interval can be as short as a few clock



(a) Effect of k_t on reliability (b) Effect of t on reliability

Fig. 2. Effect of number of read requests (k_t) and vulnerable intervals (t) on reliability of cache.

cycles or as long as several billion clock cycles. Assuming that all cache lines are error-free at time zero, we define the reliability of the cache at time T as the probability that all data words are read out correctly until T. Equivalently, the reliability of the cache is the probability that no data word has been affected by a temporal MBU in any vulnerable interval t. Considering (4) as the reliability of data words in a specific interval, the reliability of the cache at time T is according to (5):

$$R_{cache}(T) = \prod_{t=1}^{T} R_{k_t}(t) = \prod_{t=1}^{T} (1+\lambda t)^{k_t} e^{-\lambda t k_t} \tag{5}$$

It is worth noting that the unit of T and t is clock cycle. Thus, the minimum and maximum values for vulnerable interval are one clock cycle and T clock cycles, respectively.

The reliability of the cache increases by increasing the reliability of read requests to data words in each interval. More precisely, reduction in each of t and k_t parameters in (4) increases the cache reliability. Fig. 2 shows how the reliability, i.e., $R_{k_t}(t)$, is affected by t and k_t, when the error rate (λ) is 0.01. In Fig. 2(a), t is fixed to 10 and k_t varies from zero to 100. As depicted, reliability reduces by larger values of k_t. By fixing k_t to 10 and increasing t from zero to 100 in Fig. 2(b), the reliability is again reduced.

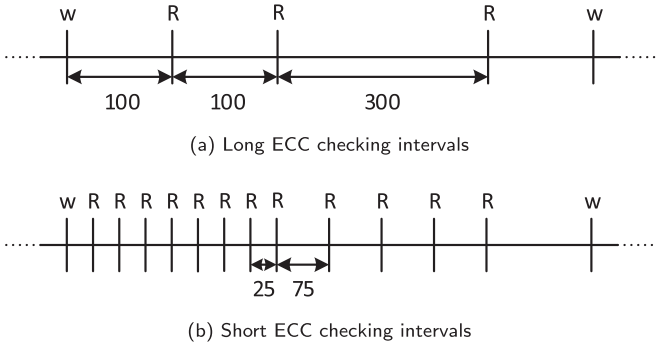
Now, we will answer the following question: *How the reliability is affected in (4) by increasing one of* t or k_t *and reducing the other one, while keeping their product* ($t \times k_t$) *fixed?* As an example for this question, Fig. 3 shows two simplified patterns of accesses to a data word in the cache. In Fig. 3(a), the vulnerable intervals (t) are 100 and 300 clock cycles. Two read accesses are performed after 100 clock cycles ($k_{100} = 2$) and one read access is performed after 300 clock cycles ($k_{300} = 1$). Replacing the parameters in (4) with the above values, reliability of these three read accesses (the occurrence probability of no temporal MBU in any of the three intervals) is calculated by (6):

$$\begin{aligned} R_a &= R_2(100) + R_1(300) \\ &= (1+100\lambda)^2 e^{-200\lambda} + (1+300\lambda)e^{-300\lambda} \end{aligned} \tag{6}$$

In Fig. 3(b), the frequency of read accesses to the data word is quadrupled. Therefore, the interval durations are reduced to 25 and 75 clock cycles, and there are eight and four read accesses for each interval, respectively. Reliability of these 12 read accesses are according to (7):

$$\begin{aligned} R_b &= R_8(25) + R_4(75) \\ &= (1+25\lambda)^8 e^{-200\lambda} + (1+75\lambda)^4 e^{-300\lambda} \end{aligned} \tag{7}$$

To answer the above-mentioned question, i.e., the effect of k_t and t on reliability, we assume that k_t and t in (4) are increased and reduced, respectively, by a factor of α ($\alpha \geq 1$). Thus, the number of accesses after each interval is increased, whereas the duration of each interval is



(a) Long ECC checking intervals

(b) Short ECC checking intervals

Fig. 3. Two examples for pattern of accesses to a data word with different ECC checking intervals.

reduced. Considering (4), the reliability in this scenario is according to (8):

$$\begin{aligned} R_{k_{t'}}(t') &= (1+\lambda t')^{k_{t'}} e^{-\lambda t' k_{t'}} \\ &= (1+\lambda \frac{t}{\alpha})^{\alpha k_t} e^{-\lambda t k_t} \end{aligned} \tag{8}$$

where t' and $k_{t'}$ are equal to t/α and αk, respectively. For every integer $\alpha \geq 1$, the inequality in (9) can be proven:

$$R_{k_{t'}}(t') \geq R_{k_t}(t) \tag{9}$$

where $R_{k_{t'}}(t')$ and $R_{k_t}(t)$ have been given in (8) and (4), respectively. To prove (9), we need to show that (10) is true:

$$(1+\lambda \frac{t}{\alpha})^{\alpha k_t} \geq (1+\lambda t)^{k_t} \quad \alpha \geq 1, \quad k_t \geq 0 \tag{10}$$

By removing k_t from both sides of the inequality in (10), it is equivalent to the well-known *Bernoulli's inequality*, which is given in (11):

$$(1+x)^r \geq 1 + rx \qquad r \geq 0, \quad x \geq -1 \tag{11}$$

By substituting r and x in (11) with α and t/α in (10), respectively, the inequalities in (9) and (10) will be proven.

To quantitatively investigate the effect of α on the reliability of the cache, Fig. 4 shows the values of $R_{k_{t'}}(t')$ calculated according to (8) for wide ranges of k_t and α. Here, k_t (the number of data words that are read after t clock cycles) varies from 1 to 2000, α varies from 1 to 50, and λt is fixed to 0.1. Fig. 4 depicts that α can significantly affect the reliability. For example, $R_{k_{t'}}(t')$ is equal to 0.009 when k_t and α is equal to 1000 and 1, respectively. $R_{k_{t'}}(t')$ is increased to 0.375, 0.605, 0.773, 0.884, and 0.877 by increasing α to 5, 10, 20, 30, and 40, respectively. On the other hand, increasing α can compensate the negative effect of increasing k_t on the reliability. As an example, $R_{k_{t'}}(t')$ is reduced from

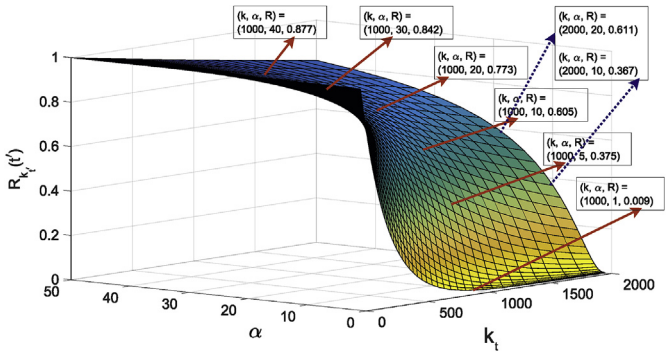


Fig. 4. Reliability of cache for different values of k_t and α ($\lambda t = 0.1$).

0.605 to 0.367 by increasing k_t from 1000 to 2000 when α is 10; and, is increased to 0.611 by increasing α to 20 when k_t remains at 2000.

The above formulations show how the number and interval of accesses to cache lines affect the reliability of the cache by affecting the occurrence probability of temporal MBUs. These formulations is not limited to caches and can also be extended to other types of memories, e.g., main memory. A well-known scheme to reduce the occurrence probability of temporal MBUs is memory scrubbing [4,18,40]. This scheme periodically reads memory lines, checks their ECC bits, and overwrites the corrected data if a correctable error is detected.

Scrubbing is mainly used in off-chip memories for preventing error accumulation in data words [18,40]. In on-chip caches, however, scrubbing is not affordable. For a long scrubbing period, the idle intervals (the interval between two consecutive requests to a word) for the majority of data words in the cache are much shorter than the scrubbing period. Thus, scrubbing cannot effectively prevent the occurrence of temporal MBUs in these data words [44]. Reducing the scrubbing period decreases the idle intervals for less frequently-accessed words, which can practically prevent the occurrence of temporal MBUs. However, reducing the scrubbing period as short as it can affect a considerable fraction of cache words can increase the memory traffic and cache energy consumption [4]. In addition, it can degrade the performance of the system if the scrubbing operations interfere with the normal cache operations [4,41].

3. The proposed CLEAR scheme

Fig. 5 shows an abstract view for the conventional architecture of a 4-way associative cache. All cache protection schemes as well as performance and energy consumption improvement schemes in the literature are designed based on this configuration, even if they are not care about it. The goal of Fig. 5 is to illustrate the novelty of this work and to better explain its operation. The key idea in this work is to take the advantage of an inherently available redundancy in the cache architecture for the purpose of reliability improvement. Fig. 5 demonstrates that inherent redundancy.

The cache operation is as follows. On a read request, the corresponding cache set is determined by *Index* field in the requested address. Then, tags of all cache ways in the set are read out to be compared with *Tag* field of the address. Simultaneously, based on *Offset* field of the address, data word and their ECC bits of all cache ways in the set are read out. In the next step, the requested data is selected by the output of the tag comparison operation and is fed to *ECC Decoder* unit. Finally, the data is checked based on its corresponding ECC bits and is sent out.

As observed, all cache ways in data array are accessed in parallel with the tag comparison operation for reading a single data word. The requested word is sent out and the other accessed words are discarded. Accessing all cache ways prior to determining the target way is performed to minimize the latency of cache read accesses. This approach

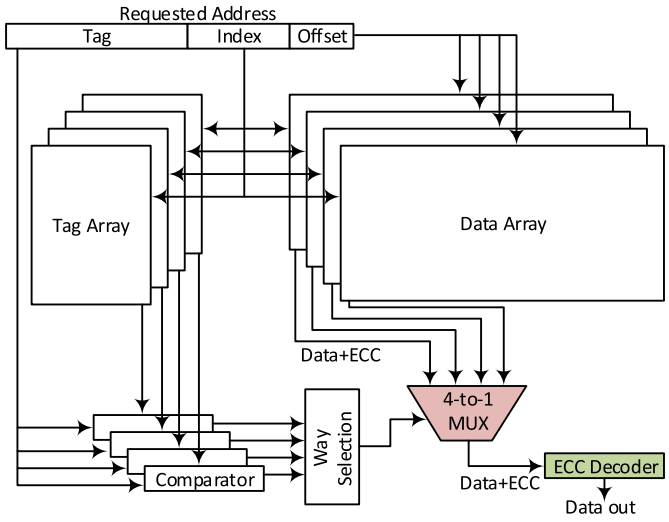


Fig. 5. Conventional cache architecture. All cache blocks in accessed set enter the MUX unit and only the block selected by the MUX unit is checked by the ECC decoder unit.

for accessing the cache is known as *fast mode* or *parallel mode* and is used in almost all L1 caches and some L2 caches [36,45,46]. This access mode is in contrast with *phased access mode* in which data array is accessed after completion of tag comparison operation [45]. Phased access mode is mainly used in last-level caches to save dynamic energy.

By exploiting the inherent redundancy in caches embedded to reduce the cache access latency, we propose a scheme to improve the reliability of caches. This scheme, so-called *Cache Line Error Accumulation Reduction* (CLEAR), reduces the vulnerable intervals of the cache words by conducting several ECC checking between two consecutive read requests to a word. The key idea in CLEAR is to check all words that have been read out for a read request instead of checking only the requested word. CLEAR conducts an *early error checking* for all cache ways by exploiting the inherently available externally invisible accesses.

Fig. 6 shows the cache architecture in CLEAR. As depicted, CLEAR makes a minor modification in conventional cache architecture by reordering the ECC decoder unit and the MUX unit that selects the requested words between all words. The ECC decoder unit is replicated

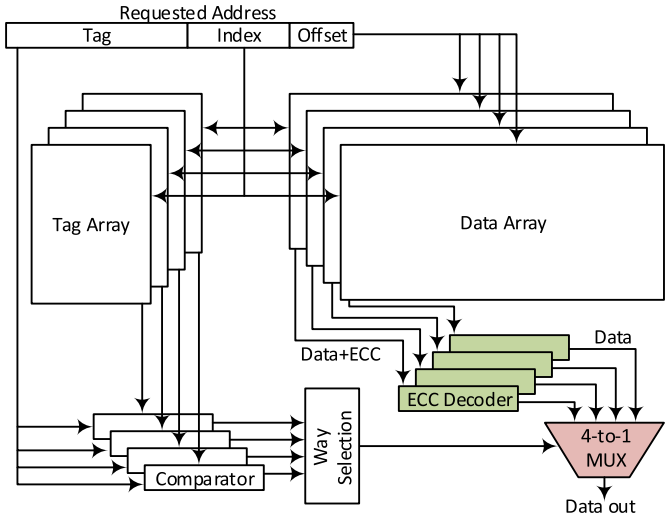


Fig. 6. Cache architecture in CLEAR. The locations of the ECC decoder and the MUX units are exchanged and all blocks in the accessed set are checked by the ECC decoder unit before sending them to the MUX unit.

Table 1
System Configuration.

Core	2 GHz, Out-of-Order, 4-issue superscalar
L1 D-cache	32-Kbyte, 4-way associative, LRU, 64-byte cache line, write-back, 2-cycle read, 2-cycle write
L1 I-cache	32-Kbyte, 4-way associative, LRU, 64-byte cache line, 2-cycle read, 2-cycle write
L2 cache	256-Kbyte, 8-way associative, LRU, 64-byte cache line, write-back, 10-cycle read, 10-cycle write

for all cache ways to check the words in parallel. For a read request, data words in all cache ways are first checked by their own ECC decoder unit and then fed to the MUX unit. Therefore, unlike conventional cache architecture in which data word in Most-Recently Used (MRU) cache way is checked, CLEAR conducts error checking for MRU word as well as early error checking for words in other cache ways. By checking all cache ways in each read access, the words can be periodically checked several times before sending them out.

Several ECC checking before requesting the words not only reduces the occurrence probability of temporal MBUs, but also provides the opportunity of an *early error detection* in cache words. Early error detection is mainly beneficial for parity-protected caches, in which correcting the errors requires recovery procedures. When errors are detected before requesting the data words, there may be enough time to perform the error correction operations beforehand.

Instruction caches and write-through data caches are commonly protected by parity check codes. Errors in these caches are corrected by invalidating the erroneous cache lines and refetching the data from lower memory hierarchy. Early error detection makes it possible to conduct these recovery operations in background and eliminate the error recovery overhead when data is requested. Write-back data caches are also protected by parity check codes in some processors and errors are reported to the system for error handling. Early error detection activates the error handling operations prior to accessing the erroneous data, which can eliminate or reduce the error recovery overheads. Considering parity-protected caches, parity check is able to detect an odd number of bit flips in a word and an erroneous cache word affected by temporal MBU may not be detectable by parity check. CLEAR increases the reliability of parity-protected caches by reducing the occurrence probability of temporal MBUs.

Besides the errors that can be corrected by cache protection schemes, Detectable Unrecoverable Errors (DUEs) are also probable in caches [40,47,48]. DUEs can affect the final output and result in process-kill or system-kill [47]. In CLEAR, the system is informed about the existence of such errors much earlier than the errors are activated. This early error detection helps the system to get ready for an erroneous situation. The system may be able to prevent the error activation, safely reset, or identify and re-execute the erroneous process beforehand.

4. Simulation setup and results

This section evaluates the reliability of the cache in CLEAR scheme and discusses its effects on energy consumption, area, and performance of the cache. The gem5 full-system simulator [26] is used to model an out-of-order processor and its peripherals. Table 1 summarizes the system configuration. SPEC CPU2006 benchmark suite [43] is used as the workloads. The workloads are executed for three billion instructions after skipping the warm-up phase. It is assumed that CLEAR is employed in data cache and the results are compared with the baseline, which is the conventional data cache depicted in Fig. 5. The energy consumption parameters for the cache is extracted from CACTI 6.5 cache model [49] for 65-nm technology node.

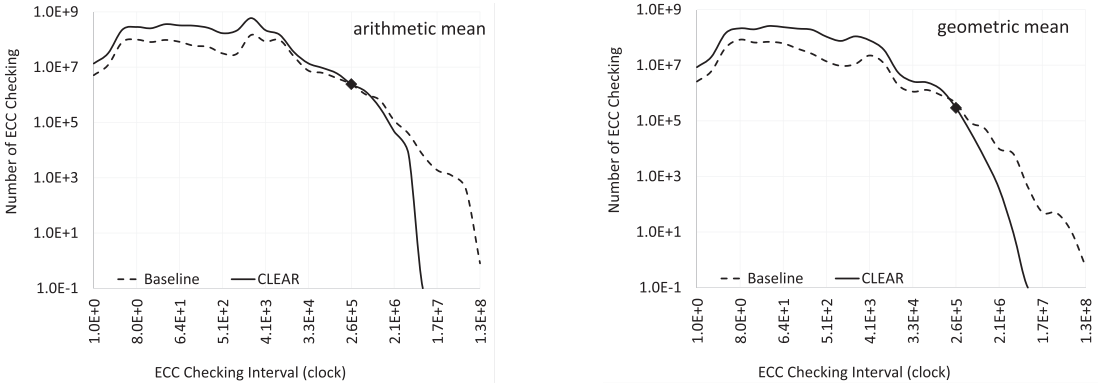
4.1. Reliability analysis

4.1.1. ECC checking intervals in CLEAR

CLEAR improves the reliability of the cache by conducting more frequent ECC checking for data words in the cache, which reduces the ECC checking interval for each word. In the following, we first investigate the effect of CLEAR on ECC checking intervals. Then, the results for reliability and mean time to failure (MTTF) are presented.

Fig. 7 shows all ECC checking intervals observed in the cache and the number of words checked for each interval. For a better visibility, the intervals are approximated by power-of-two values and the number of words accessed in intervals between 2^{n-1} and 2^n are accumulated in 2^n. For example, the number of ECC checking in interval 512 includes the total number of ECC checking in intervals 257, 258, 259, …, 511, and 512. Both x-axis and y-axis are shown in logarithmic scale.

Fig. 7(a) depicts the number of ECC checking in the intervals averaged for all workloads. Considering point $262,144$ ($= 2^{18}$) indicated in the chart, the number of ECC checking in CLEAR is larger than that in the baseline for intervals shorter than this value. However, the number of ECC checking in intervals larger than 2^{18} is significantly reduced in CLEAR. The largest interval in the baseline is equal to $134,217,728$ ($= 2^{27}$), which is reduced to $8,388,608$ ($= 2^{23}$) in CLEAR. The same trend is observed in Fig. 7(b), which shows geometric mean of the ECC checking intervals for all workloads. Considering geometric mean, the CLEAR curve crosses the baseline curve again at point $262,144$ ($= 2^{18}$).



(a) Average for all workloads (b) Geometric mean for all workloads

Fig. 7. Number of ECC checking for each ECC checking interval. Both x-axis and y-axis are in logarithmic scale.

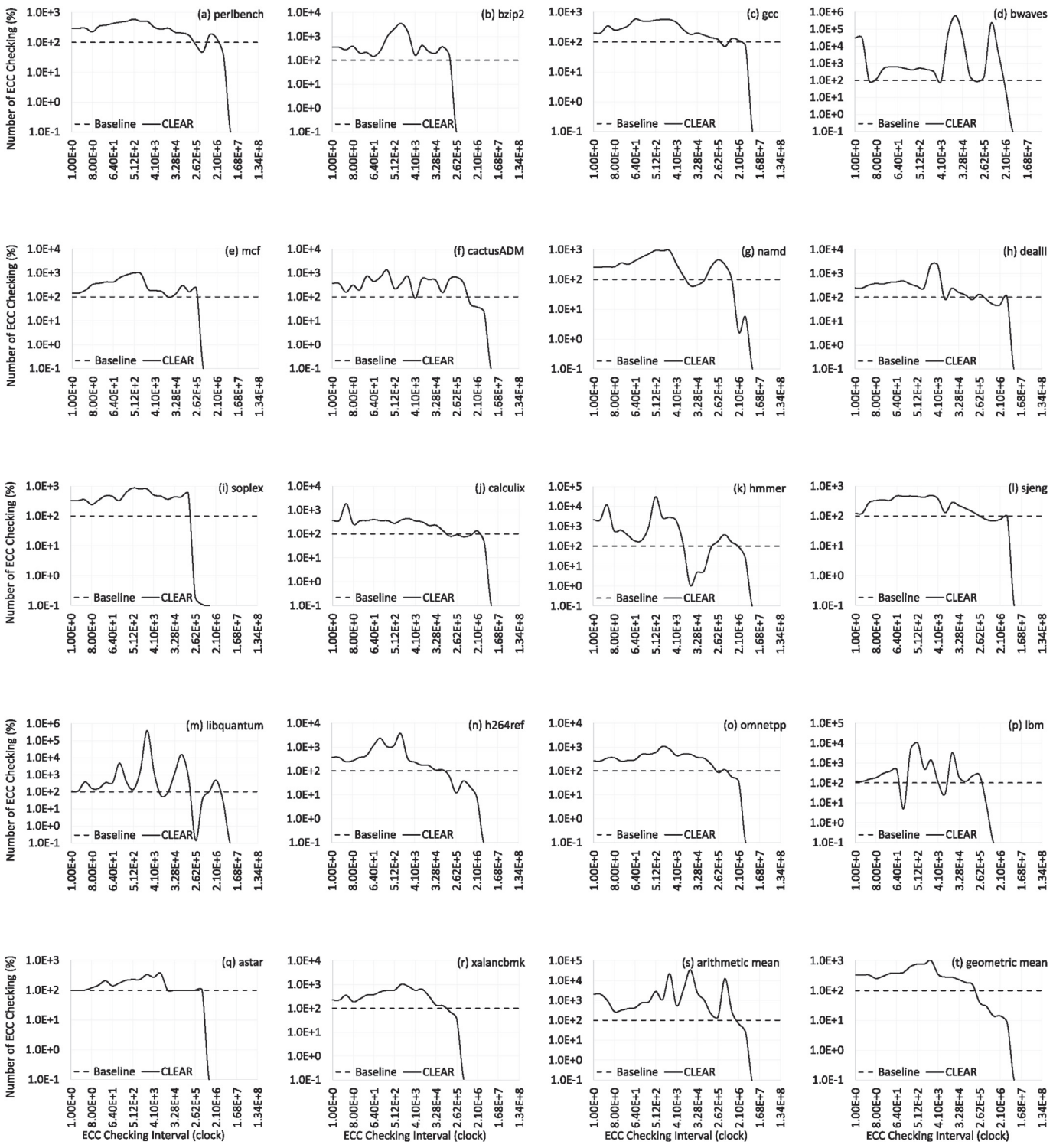


Fig. 8. Number of ECC checking for all intervals in CLEAR normalized to the baseline for different workloads.

These observations show that CLEAR shortens the ECC checking intervals for all data words by increasing the number of ECC checking. From another perspective, CLEAR pushes the baseline curve from the right to the left in Fig. 7(a) and (b), which makes the curve taller and narrower. These changes in the curve increases the reliability of the cache.

To further investigate how CLEAR affects the ECC checking, Fig. 8 shows the number of ECC checking in all intervals for each workload separately. For each interval, we based the number of ECC checking in the baseline to 100 and the number of ECC checking in CLEAR is normalized to the baseline. Fig. 8 also includes the arithmetic and geometric mean of all 18 workloads. For better visibility, both x-axis and y-axis in all charts are in logarithmic scale; and, we limited the minimum value in y-axis to 0.1 and all values smaller than 0.1 (including zero) are rounded to 0.1.

According to Fig. 8, the largest interval in the baseline is $134,217,728$ ($= 2^{27}$) for all workloads, which indicates that there are some data words in the cache that their ECC checking interval is larger

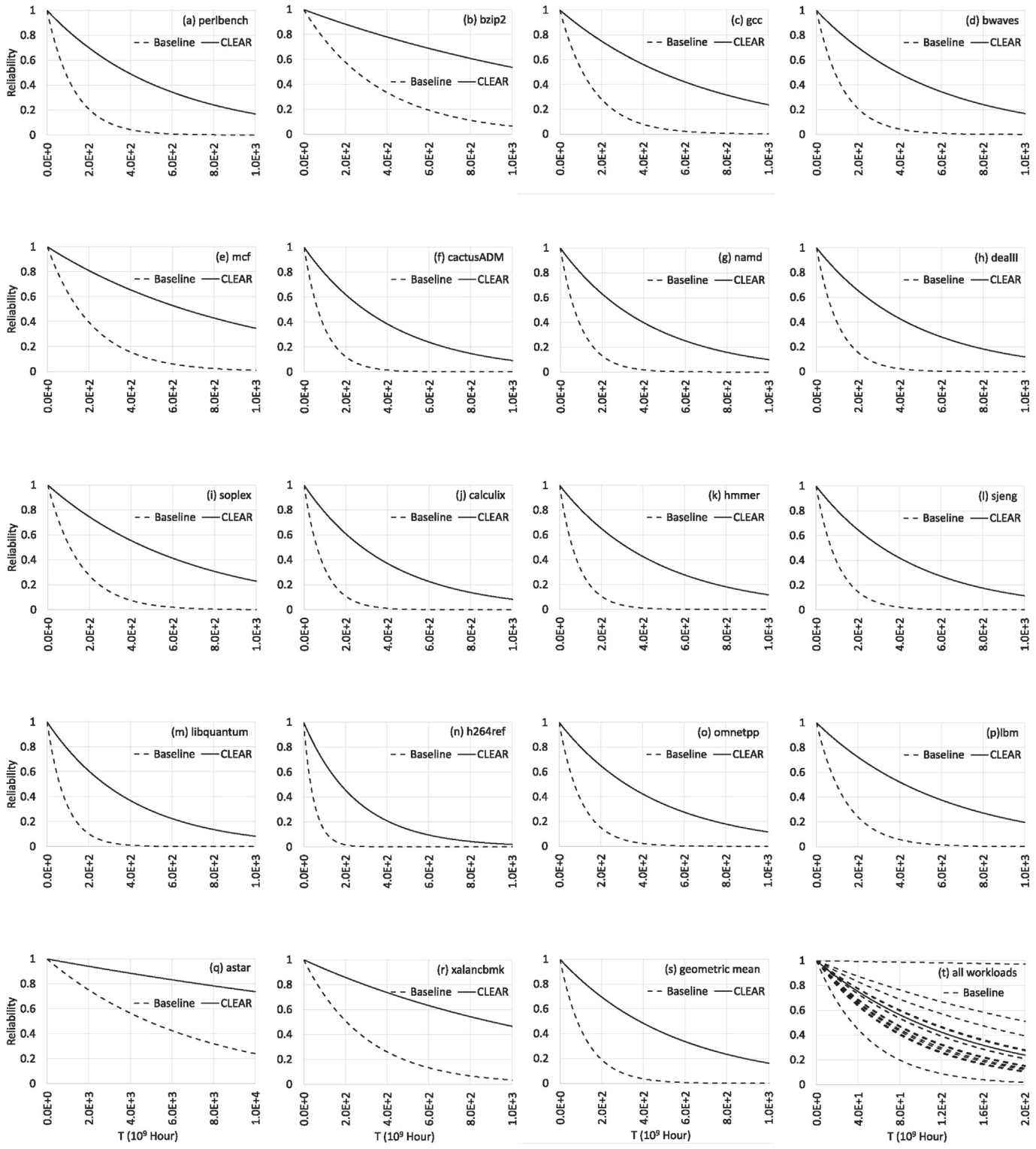


Fig. 9. Reliability of cache in baseline and CLEAR for different workloads.

than $67,108,864$ $(= 2^{26})$ clock cycles after executing three billion instructions. The largest ECC checking interval in CLEAR is $8,388,608$ $(= 2^{23})$, which is observed in nine workloads, i.e., (a) *perlbench*, (c) *gcc*, (f) *cactusADM*, (g) *namd*, (h) *dealII*, (j) *calculix*, (k) *hmmer*, (l) *sjeng*, and (m) *libquantum*. The largest interval in (d) *bwaves*, (n) *h264ref*, and (o) *omnetpp* is $4,194,304$ $(= 2^{22})$. This value in (i) *soplex*, (p) *lbm*, and (q) *astar* is reduced to $1,048,576$ $(= 2^{20})$, in (e) *mcf* and (r) *xalancbmk* is reduced to $524,288$ $(= 2^{19})$, and in (b) *bzip2* is reduced to $262,144$ $(= 2^{18})$. Based on these observations, CLEAR reduces the largest ECC checking interval by 16× in the worst case and by 512× in the best case.

As another observation, CLEAR curve in all workloads crosses the baseline line before or at point $2,097,152$ $(= 2^{21})$, and falls rapidly after this point. Considering all workloads, the CLEAR curve lies below the baseline after points $1,048,576$ $(= 2^{20})$ and $262,144$ $(= 2^{18})$ for (s) arithmetic mean and (t) geometric mean charts, respectively. These

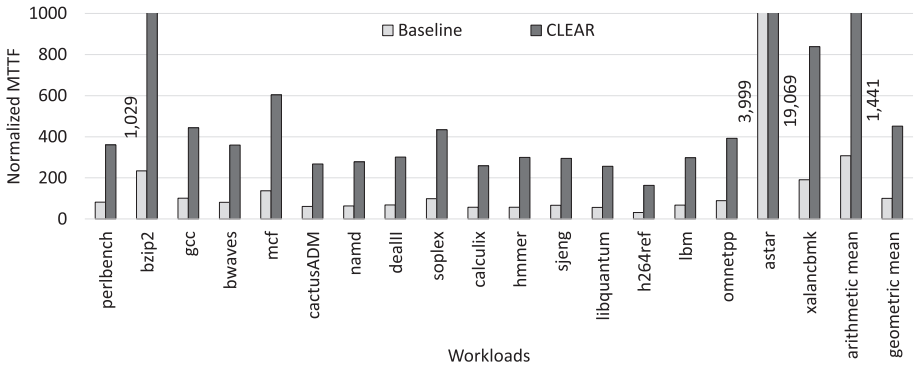


Fig. 10. Mean time to failure (MTTF) in baseline cache and CLEAR.

observations indicate that CLEAR effectively reduces the number of words checked in intervals longer than $2,097,152$ ($= 2^{21}$).

4.1.2. Cache reliability in CLEAR

We calculate the reliability of the cache according to the formulations presented in section 2. As discussed, the ECC checking intervals and the number of words checked in each interval affect the reliability of the cache. In Section 4.1.1, we investigated how CLEAR changes these two parameters.

Eq. (5) is used to calculate the reliability. The simulation time is scaled to represent different values of parameter T in (5). The ECC checking intervals, i.e., parameter t in (5), are assumed to be, for all values of T, the same as values extracted during the simulation. The values extracted for the number of words checked in each interval, i.e., parameter k_t in (5), are scaled by the ratio of T to simulation time. Parameter λ for each memory cell is optimistically assumed to be equal to the intrinsic SEU rate for SRAMs, i.e., 1.150 SEUs per 10^9 h [9].

Fig. 9 shows the reliability of the cache for all workloads as well as the geometric mean reliability. The results in Fig. 9 are of importance from two aspects. First, CLEAR improves the reliability in all workloads by almost the same ratio. Second, the reliability of the cache is highly affected by the workload behaviors. For example, reliability of the baseline in most workloads is near zero (less than 0.01) at the end of x-axis, whereas the curve of (b) *bzip2*, (q) *astar*, and (r) *xalancbmk* are above 0.03. More specifically, reliability in *astar* is significantly higher than other workloads. Note that the x-axis in *astar* is by ten times wider than that in other workloads. For better visualization of reliability variation in different workloads, the reliability curves of the baseline cache in all workloads for a shorter operation time are also depicted in Fig. 9(t).

The results in Fig. 9 indicate that, although the occurrence probability of temporal MBU in the cache is highly workload-dependent, the reliability improvement provided by CELAR is not affected by workload behaviors. This statement will be confirmed in the following by investigating the effect of CLEAR on the MTTF of the cache. The results show that despite large variations in MTTFs of the workloads, CLEAR increases the MTTFs in all workloads with almost the same ratio.

Fig. 10 shows the MTTF in the baseline and CLEAR for all workloads. We scale the geometric mean MTTF in the baseline to 100 and normalize other MTTFs to it. In the baseline, the MTTF varies from 32 in *h264ref* to $3,999$ in *astar*, which indicates that the variation in MTTFs is more than 100×. MTTF in *bzip2* and *xalancbmk* is as high as 234 and 190, respectively, and in *cactusADM* and *hmmer* is as low as 61 and 57, respectively. Considering MTTFs in CLEAR, all values are increased by a factor between 4.39 and 5.25. The geometric mean in CLEAR shows that this scheme extends the MTTF by 4.52× (or 452%).

4.2. Overheads

The latency of the cache is not increased due to the modification made by CLEAR. This issue is investigated in the following. The cache access time in the baseline cache, shown in Fig. 5, is according to (12):

$$T_{baseline} = \max\{(t_{tag\_access} + t_{tag\_compare}), t_{data\_access}\} + t_{mux} + t_{ECC\_decoder} \quad (12)$$

And, as already has been shown in Fig. 6, CLEAR swaps the MUX unit and ECC decoder unit in read path of the cache. The cache access time in CLEAR is equal to (13):

$$T_{CLEAR} = \max\{(t_{tag\_access} + t_{tag\_compare}), \times (t_{data\_access} + t_{ECC\_decoder})\} + t_{mux} \quad (13)$$

Considering (12) and (13), one of the three following scenarios is probable:

$$t_{tag\_access} + t_{tag\_compare} < t_{data\_access} \quad (14)$$

$$t_{data\_access} < t_{tag\_access} + t_{tag\_compare} < t_{data\_access} + t_{ECC\_decoder} \quad (15)$$

$$t_{data\_access} + t_{ECC\_decoder} < t_{tag\_access} + t_{tag\_compare} \quad (16)$$

In the case of (14), the cache access latency in CLEAR is the same as that of baseline and is equal to (17):

$$T_{baseline} = T_{CLEAR} = t_{data\_access} + t_{mux} + t_{ECC\_decoder} \quad (17)$$

In the case of (15), the cache access latency in the baseline and CLEAR is equal to (18) and (19), respectively:

$$T_{baseline} = t_{tag\_access} + t_{tag\_compare} + t_{mux} + t_{ECC\_decoder} \quad (18)$$

$$T_{CLEAR} = t_{data\_access} + t_{mux} + t_{ECC\_decoder} \quad (19)$$

In this case, the cache latency in CLEAR is smaller than that in the baseline. In the case of (16), the cache latency in the baseline is again equal to (18), and in the CLEAR is equal to (20), which is again smaller than that in the baseline:

$$T_{CLEAR} = t_{tag\_access} + t_{tag\_compare} + t_{mux} \quad (20)$$

The above calculations show that the cache access latency in CLEAR is smaller than or, in the worst case, equal to that in the baseline. Therefore, CLEAR imposes no performance overhead to the system. On the other hand, ECC decoding is probable to be done in different pipe stages. In this case, swapping the MUX and ECC decoding operations only reorders the different cache pipe stages and the total cache access time remains intact, as in single cycle cache access.

Considering the cache area, CLEAR adds three ECC decoder units to the cache. Our evaluations show that the area of an ECC decoder/encoder unit is about 0.00336 mm^2 and the cache area is

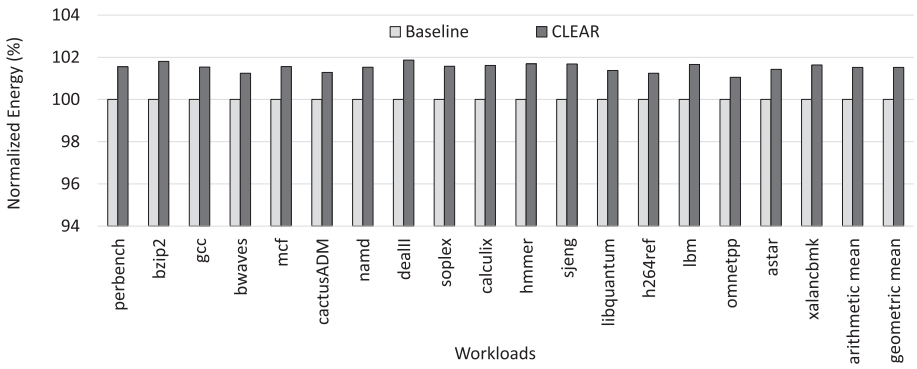


Fig. 11. Cache energy consumption in CLEAR normalized to baseline.

about 0.76 mm^2. Therefore, the area overhead of CLEAR is less than 2%.

CLEAR may increase the energy consumption in the cache by conducting a higher number of decoding operations. One decoding operation is required for each read access to the baseline cache. In CLEAR, on the other hand, four ECC decoder units perform four decoding operations in parallel.

Fig. 11 shows the cache energy consumption in CLEAR normalized to the baseline. The results depict that CLEAR increases the energy consumption by about 1.52%, on average. The minimum and maximum values for energy consumption overhead are 1.24% in *bwaves* and 1.86% in *dealII*, respectively.

5. Conclusions

Accumulation of SEUs in SRAM caches over time is a major reliability challenge, because conventional ECCs are unable to correct these temporal MBUs. This paper proposes CLEAR scheme to reduce the probability of SEUs accumulation in cache lines. CLEAR conducts the ECC checking not only for the requested data word, but also for all other words that have been read out in parallel. By conducting more frequent ECC checking, CLEAR significantly reduces the probability of temporal MBUs in cache lines. The results show that MTTF of the cache is increased by about 452% in CLEAR compared with the conventional architecture. CLEAR has no impact on the cache latency and increases the area and energy consumption by less than 2% and about 1.5%, respectively. CLEAR not only increases the reliability of the cache when cache scrubbing is not used, but also can be used for widening the scrubbing interval without reliability reduction to reduce scrubbing overhead. Moreover, it can reduce/eliminate the error recovery latency by detecting errors beforehand.

Acknowledge

This work has been partially supported by Iran's National Elites Foundation.

References

[1] J.A. Martinez, J.A. Maestro, P. Reviriego, Evaluating the impact of the instruction set on microprocessor reliability to soft errors, IEEE Trans. Device Mater. Reliab. 18 (1) (2018) 70–79.
[2] H. Liu, M. Cotter, S. Datta, V. Narayanan, Soft-error performance evaluation on emerging low power devices, IEEE Trans. Device Mater. Reliab. 14 (2) (2014) 732–741.
[3] Y. Ko, R. Jeyapaul, Y. Kim, K. Lee, A. Shrivastava, Guidelines to design parity protected write-back L1 data cache, in: Proceedings of the Design Automation Conference (DAC), 2015, pp. 1–6.
[4] S. Lee, S. Baeg, P. Reviriego, Memory reliability model for accumulated and clustered soft errors, IEEE Trans. Nucl. Sci. 58 (5) (2011) 2483–2492.
[5] S. Wang, J. Hu, S.G. Ziavras, On the characterization and optimization of on-chip cache reliability against soft errors, IEEE Trans. Comput. 58 (9) (2009) 1171–1184.
[6] A. Neale, M. Sachdev, A new SEC-DED error correction code subclass for adjacent MBU tolerance in embedded memory, IEEE Trans. Device Mater. Reliab. 13 (1) (2013) 223–230.
[7] Y.-P. Fang, A.S. Oates, Thermal neutron-induced soft errors in advanced memory and logic devices, IEEE Trans. Device Mater. Reliab. 14 (1) (2014) 583–586.
[8] G. Torrens, S.A. Bota, B. Alorda, J. Segura, An experimental approach to accurate alpha-SER modeling and optimization through design parameters in 6T SRAM cells for deep-nanometer CMOS, IEEE Trans. Device Mater. Reliab. 14 (4) (2014) 1013–1021.
[9] J. Suh, M. Annavaram, M. Dubois, Macau: a Markov model for reliability evaluations of caches under single-bit and multi-bit upsets, in: Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2012, pp. 1–12.
[10] D. Rossi, V. Tenentes, S.M. Reddy, B.M. Al-Hashimi, A. Brown, Exploiting aging benefits for the design of reliable drowsy cache memories, IEEE Trans. Comput. Aided Des. Integr Circ. Syst. 37 (7) (2018) 1345–1357.
[11] M. Gottscho, I. Alam, C. Schoeny, L. Dolecek, P. Gupta, Low-cost memory fault tolerance for iot devices, ACM Trans. Embed. Comput. Syst. 16 (5s) (2017) 128 1128:25.
[12] A. Subramaniyan, S. Rehman, M. Shafique, A. Kumar, J. Henkel, Soft error-aware architectural exploration for designing reliability adaptive cache hierarchies in multi-cores, in: Proceedings of the Conference on Design, Automation & Test in Europe (DATE), 2017, pp. 37–42.
[13] 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 Trans. Embed. Comput. Syst. 15 (4) (2016) 67 167:26.
[14] M. Manoochehri, M. Annavaram, M. Dubois, Extremely low cost error protection with correctable parity protected cache, IEEE Trans. Comput. 63 (10) (2014) 2431–2444.
[15] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, R. Gupta, Reliability-aware data placement for heterogeneous memory architecture, in: Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2018, pp. 583–595.
[16] J. Das, S. Ghosh, Energy barrier model of SRAM for improved energy and error rates, IEEE Trans. Circ. Syst.-I (TCAS-I) 61 (8) (2014) 2299–2308.
[17] F.R. Rosa, R. Brum, G. Wirth, F. Kastensmidt, L. Ost, R. Reis, Impact of dynamic voltage scaling and thermal factors on SRAM reliability, Elsevier J. Microelectron. Reliab. (MR) 55 (9) (2015) 1486–1490.
[18] H. Wang, Y. Wang, ISS: an iterative scrubbing strategy for improving memory reliability against MBU, in: Proceedings of the International Conference on Human Centered Computing (HCC), 2016, pp. 420–431.
[19] D. Kim, L. Milor, A methodology for estimating memory lifetime using a system-level accelerated life test and error-correcting codes, in: Proceedings of the IEEE VLSI Test Symposium (VTS), 2017, pp. 1–6.
[20] Y.-W. Chiu, Y.-H. Hu, M.-H. Tu, J.-K. Zhao, Y.-H. Chu, S.-J. Jou, C.-T. Chuang, 40 nm bit-interleaving 12T subthreshold SRAM with data-aware write-assist, IEEE Trans. Circ. Syst.-I (TCAS-I) 61 (9) (2014) 2578–2585.
[21] Y. Yang, H. Jeong, S.C. Song, J. Wang, G. Yeap, S.-O. Jung, Single bit-line 7T SRAM cell for near-threshold voltage operation with enhanced performance and energy in 14 nm FinFET technology, IEEE Trans. Circ. Syst.-I (TCAS-I) 63 (7) (2016) 1023–1032.
[22] S. Pal, A. Islam, 9-T SRAM cell for reliable ultralow-power applications and solving multibit soft-error issue, IEEE Trans. Device Mater. Reliab. 16 (2) (2016) 172–182.
[23] S. Ganapathy, J. Kalamatianos, K. Kasprak, S. Raasch, On characterizing near-threshold sram failures in finfet technology, in: Proceedings of the 54th Annual Design Automation Conference (DAC), ACM, 2017, p. 53. 153:6.
[24] J. Tonfat, L. Tambara, A. Santos, F.L. Kastensmidt, Soft error susceptibility analysis methodology of hls designs in sram-based fpgas, Microprocess. Microsyst. 51 (2017) 209–219.
[25] A.M. Keller, M.J. Wirthlin, Benefits of complementary seu mitigation for the leon3 soft processor on sram-based fpgas, IEEE Trans. Nucl. Sci. 64 (1) (2017) 519–528.
[26] W. Choi, J. Park, A charge-recycling assist technique for reliable and low power SRAM design, IEEE Trans. Circ. Syst.-I (TCAS-I) 63 (8) (2016) 1164–1175.

[27] A. Bosser, V. Gupta, G. Tsiligiannis, C. Frost, A. Zadeh, A. Javanainen, H. Puchner, F. Saign, A. Virtanen, F. Wrobeland, L. Dilillo, Methodologies for the statistical analysis of memory response to radiation, IEEE Trans. Nucl. Sci. 63 (4) (2016) 2122–2128.
[28] A. Neale, M. Sachdev, Neutron radiation induced soft error rates for an adjacent-ECC protected SRAM in 28 nm CMOS, IEEE Trans. Nucl. Sci. 63 (3) (2016) 1912–1917.
[29] H. Farbeh, F. Mozafari, M. Zabihi, S.G. Miremadi, Raw-tag: replicating in altered cache ways for correcting multiple-bit errors in tag array, IEEE Trans. Dependable Secure Comput. (2017).
[30] S.G. Ghaemi, I. Ahmadpour, M. Ardebili, H. Farbeh, SMARTag: error correction in cache tag array by exploiting address locality, in: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2018, pp. 1658–1663.
[31] H. Farbeh, H. Kim, S.G. Miremadi, S. Kim, Floating-ecc: dynamic repositioning of error correcting code bits for extending the lifetime of stt-ram caches, IEEE Trans. Comput. 65 (12) (2016) 3661–3675.
[32] J. Hong, S. Kim, Smart ecc allocation cache utilizing cache data space, IEEE Trans. Comput. 66 (2) (2017) 368–374.
[33] P. Reviriego, S. Pontarelli, A. Evans, J.A. Maestro, A class of SEC-DED-DAEC codes derived from orthogonal Latin square codes, IEEE Trans. Very Large Scale Integr. Syst. 23 (5) (2015) 968–972.
[34] H. Farbeh, M. Fazeli, F. Khosravi, S.G. Miremadi, Memory mapped spm: protecting instruction scratchpad memory in embedded systems against soft errors, in: 2012 Ninth European Dependable Computing Conference, 2012, pp. 218–226.
[35] Z. Azad, H. Farbeh, A.M.H. Monazzah, S.G. Miremadi, Aware: adaptive way allocation for reconfigurable eccs to protect write errors in stt-ram caches, IEEE Trans. Emerg. Top. Comput. (2017).
[36] H. Farbeh, S.G. Miremadi, PSP-cache: a low-cost fault-tolerant cache memory architecture, in: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2014, pp. 1–4.
[37] L. Delshadtehrani, H. Farbeh, S.G. Miremadi, In-scratchpad memory replication: protecting scratchpad memories in multicore embedded systems against soft errors, ACM Trans. Des. Autom. Electron. Syst. 20 (4) (2015) 61.
[38] H. Farbeh, N.S. Mirzadeh, N.F. Ghalaty, S.G. Miremadi, M. Fazeli, H. Asadi, A cache-assisted scratchpad memory for multiple-bit-error correction, IEEE Trans. Very Large Scale Integr. Syst. 24 (11) (2016) 1–14.
[39] S. Cha, H. Yoon, Single-error-correction and double-adjacent-error-correction code for simultaneous testing of data bit and check bit arrays in memories, IEEE Trans. Device Mater. Reliab. 14 (1) (2014) 529–535.
[40] S.S. Mukherjee, J. Emer, T. Fossum, S.K. Reinhardt, Cache scrubbing in microprocessors: myth or necessity, in: Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2004, pp. 37–42.
[41] J. Hong, J. Kim, S. Kim, Exploiting same tag bits to improve the reliability of the cache memories, IEEE Trans. Very Large Scale Integr. Syst. 23 (2) (2015) 254–265.
[42] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, D.A. Wood, The gem5 simulator, Comput. Architect. News 39 (2) (2011) 1–7.
[43] J.L. Henning, SPEC CPU2006 benchmark descriptions, Comput. Architect. News 34 (4) (2006) 1–17.
[44] L.D. Hung, M. Goshima, S. Sakai, Mitigating soft errors in highly associative cache with CAM-based tag, in: Proceedings on the IEEE International Conference on Computer Design (ICCD), 2005, pp. 342–347.
[45] J. Dai, L. Wang, An energy-efficient L2 cache architecture using way tag information under write-through policy, IEEE Trans. Very Large Scale Integr. Syst. 21 (1) (2013) 102–112.
[46] E. Cheshmikhani, H. Farbeh, H. Asadi, Enhancing reliability of STT-MRAM caches by eliminating read disturbance accumulation, in: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2019, pp. 854–859.
[47] S.S. Mukherjee, J. Emer, S.K. Reinhardt, The soft error problem: an architectural perspective, in: Proceedings of the IEEE International Symposium on High-Performance Comp Architecture (HPCA), 2005, pp. 1–12.
[48] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, P. Gupta, Measuring the impact of memory errors on application performance, IEEE Comput. Archit. Lett. 16 (1) (2017) 51–55.
[49] N. Muralimanohar, R. Balasubramonian, N. Jouppi, Tech. Rep. HPL-2009-85, in: CACTI 6.0: A Tool to Model Large Caches, 2009.