This is a postprint version of the following published document:

Liu, S., Reviriego, P., Lombardi, F. (2021). Protection of Associative Memories Using Combined Tag and Data Parity (CTDP). *IEEE Transactions on Nanotechnology*, 20, 1–9.

DOI: <https://doi.org/10.1109/TNANO.2020.3042114>

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Protection of Associative Memories Using Combined Tag and Data Parity (CTDP)

Shanshan Liu, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, and Fabrizio Lombardi, *Fellow, IEEE*

Abstract—As emerging memories are utilized in processors as main memory, they must also coexist with CMOS memories; for instance SRAMs, are used to implement smaller, but faster associative memories. These hybrid designs exploit the advantages of both types of memories to achieve better performance. For some applications, the improvement in performance for on-chip associative memories is crucial for the overall computing system. For CMOS memories, soft errors are a major concern because they flip bits and can lead to data corruption and even a system failure. Error Detection and Correction Codes (EDCCs) are commonly used to protect memories against soft errors. In associative memories, an entry is formed by a tag and its associated data (or value). EDCCs are typically used to separately protect the tag and the data. This paper considers the protection of associative memories in which false negatives do not cause a failure. A Combined Tag and Data Parity (CTDP) protection scheme is proposed. This new approach utilizes a single parity bit per entry, so it reduces the number of parity bits needed to protect an entry as well as the memory size. The proposed scheme also reduces the complexity of read operations; it incurs the lowest circuit overhead for the protection circuitry in terms of area, delay and power consumption when compared to other schemes found in the technical literature. This makes the proposed scheme germane to associative memories used in hybrid designs that combine SRAMs and emerging memories. The extension of the proposed scheme to stronger codes is also discussed.

Index Terms—Associative memories, soft error, error detection and correction codes

I. INTRODUCTION

THE protection of storage systems such as memories against soft errors (caused by phenomena such as Single Event Upsets (SEUs)) has been widely studied in the technical literature; these errors can lead to data corruption and in some cases even to a system failure [1]–[4]. Error detection and correction codes (EDCCs) initially applicable to errors in data transmission are also commonly used to protect memories (including both conventional CMOS memories [5], [6] and emerging next-generation memories based on alternative technologies [7], [8]) in the presence of soft errors. Different

from protecting data in communication channels, EDCCs with a smaller number of parity bits and a lower complexity for decoding are better suitable to memory application. Memories represent a significant part of the area of an integrated circuit such as modern processors [9]; the number of parity bits stored on each word impacts the memory overhead for protecting stored data. Moreover, decoding affects latency during the data read operation.

Associative memories are widely used in many applications to efficiently retrieve data. For example, a widely used associative memory is cache, which stores frequently used data to increase the throughput of processors [10]. Some associative memories require fast and high frequent data accesses (e.g., L1 caches and Transaction Lookaside Buffers (TLBs)); thus even though emerging techniques are very promising in memory implementation to improve storage density, they are usually employed for main memory and/or the larger caches (e.g., L2 and L3 caches). Instead, SRAMs are still a better solution for the faster and smaller associative memories like L1 caches and TLBs in general because most emerging memories are not sufficiently fast and reliability may be a problem (e.g., due to the long write latency and limited write endurance) [11]–[13]. Therefore, systems/processors that require high performance usually adopt a “hybrid” memory configuration (e.g., deploying emerging memories as main memory and SRAMs as caches [14]); in this case, EDCCs with different error control features are utilized for the different memories to meet both specific performance and protection requirements. For such hybrid designs, associative memories are important because they must compensate some of the disadvantages of emerging memories, such as lower speed or reduced endurance for read or write operations. Therefore, the overhead reduction for protecting associative memories in terms of speed, area and power is relevant for these hybrid systems.

In associative memories, each entry has at least a so-called tag or key that identifies the address to the corresponding data (also denoted as value). In the process of retrieving data, the incoming tag is checked with several entries (in parallel in most cases [9]). If there is a match between the incoming tag and a stored tag, the data associated with the matching tag is accessed. An error in an associative memory can have different effects depending on whether the error affects the tag, or the data [15]. If the tag is corrupted, there are two scenarios, false positives and false negatives. False negatives occur when an incoming tag should have matched a stored tag; however, it is not matched because it has been altered by the error. In such case, a miss is said to occur (for caches, the corresponding data is then obtained from main memory). There is no data corruption as

This work was supported by the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Science and Innovation and by the Madrid Community research project TAPIR-CM grant no. P2018/TCS-4496.

S. Liu and F. Lombardi are with Northeastern University, Dept. of ECE, Boston, MA 02115, USA (email: sslu@ece.neu.edu, lombardi@ece.neu.edu)

P. Reviriego is with Universidad Carlos III de Madrid, Av. Universidad 30, Leganés, Madrid, Spain (email: revirieg@it.uc3m.es).

Copyright (c) 2020 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

long as the data stored in the entry whose tag was affected by the error, has not been modified. An error is likely to cause a false negative because the data stored in the memory is frequently accessed. A false positive occurs when an incoming tag matches a stored tag because it has been modified by an error. In this case, an incorrect value is returned, and this may lead to a system error. For an error to induce a false positive, an access to a tag that differs in only one bit from the tag that has suffered the error must occur. This is generally less likely than a false negative and depends on the memory workload. An error on the data will also cause data corruption if the affected entry is used before it gets eventually overwritten.

In particular, for some associative memories, the only errors causing failure are false positives on tags as well as errors on data [15]. The protection of these types of memories has significant practical interest because some associative memories are intrinsically read-only, such as read-only (e.g., L1 Instruction caches (ICs) and TLBs) and write-through caches (e.g., some L1 data caches). False negatives on such memories only cause a miss that can degrade performance but it does not affect the correctness of the results (while for write-back caches, false negatives can also be an issue because the modified data may be overwritten due to the false negative). In most cases, errors are rare events and thus the impact on performance caused by false negatives is negligible. For example, radiation-induced soft errors occur in a time scale of at least days and thus, the cache misses induced by the errors have no impact on performance. This observation has led to the use of error detection codes that have a smaller number of parity bits and can be easily implemented to protect those types of associative memories. Particularly, for errors that affect a single bit, a commonly used scheme is to use a single parity check to implement single error detection. This adds only one parity bit to the tag and another one to the data to detect errors separately. An effective separate parity protection scheme for read-only caches, proposed in [16], can detect errors faster by comparing the encoded tags instead of checking the parity.

In this paper, a Combined Tag and Data Parity (CTDP) scheme using a single parity bit per entry is presented. It can effectively protect associative memories, in which only false positives cause a reliability issue, against single bit errors, while significantly reducing the circuitry needed for protection.

The rest of this paper is organized as follows. Section II reviews the previous separate parity protection (SPP) techniques for associative memories. The Combined Tag and Data Parity (CTDP) scheme to protect associative memories against false positives due to single bit errors is presented in Section III, in which caches are taken as the case study and evaluated in Section IV. The extension for the proposed scheme to stronger codes is discussed in Section V. More potential applications of the proposed CTDP scheme are discussed in Section VI; in the same section, the paper ends with the conclusion and future work.

II. SEPARATE PARITY PROTECTION TECHNIQUE

In traditional separate parity protection (SPP) schemes that protect associative memories, two parity bits are needed in each entry: one covers the tag bits and the other one covers the data

Fig. 1. A separate parity protected (SPP1) n -way cache.

bits. When retrieving the data, the parity bit on the tag (data) bits can detect any single bit error when searching for a matched address (outputting the associated data), thus avoiding any false positive or reading an incorrect data out.

The read operation on an n -way parity protected cache with k_1 -bit tags and k_2 -bit data is considered as example to analyze the SPP schemes. For the sake of simplicity in presentation and with no loss of generality, the Random-Access Memories (RAMs) for the tags are separate from the RAMs for the data to better illustrate the sequence of operations. In an actual implementation, the tags will likely be in a single memory structure.

Figure 1 shows the cache protected with the traditional SPP1 scheme. The index is used to select the line and the offset is used to select the word in the line that typically has more than one word (not shown in this figure for simplicity). The access to the cache first reads the position that corresponds to the incoming address; then for each way, the parity of the tag is checked against the stored parity, which is implemented by using the Parity check block in Figure 1. If there is an error, then the tag is not considered further in the process. After checking that the tags are correct, they are compared with the incoming tag. Signal 1 in Figure 1 indicates the case of “miss” or “hit” (equal to “1” for a “miss” and “0” for a “hit”). If there is a match, Signal 1 is “0” and the data that corresponds to that tag is read next; its correctness is checked by using the Parity check block. If there is no error, Signal 2 in Figure 1 (as an error signal) is “0” and the data can be read out.

As per Figure 1, the SPP1 scheme for the tags and data requires the following elements:

- A parity bit for each tag.
- A parity bit for each data.
- A parity check circuit for each way.
- A k_1 bit comparison logic for each way.
- A parity check circuit for the selected data.

An alternative scheme (denoted by SPP2) is illustrated in Figure 2; SPP2 encodes the incoming tag to obtain its parity bit and performs the comparisons in parallel and against the parity protected tag [16]. This avoids re-computing the parity on each way:

Fig. 2. Separate parity protected (SPP2) n -way cache of [16].

- If there is no match, a miss occurs (i.e., Signal 1 = "1") and the system retrieves the data from next level cache or from external memory.
- If there is a match (i.e., Signal 1 = "0"), then the data that corresponds to the matching tag is read, and its parity is checked. If the parity is correct, the data is used, otherwise a miss is generated to get the correct data.

This parity protection ensures that the execution results will be correct when single errors affect tags or data. For occurrence scenarios of multiple errors, stronger error correction codes or interleaved parity bits can be used [17], [18]. However, an additional overhead is incurred in these schemes because more parity bits are needed for the tags and data in each line.

As per Figure 2, the SPP2 scheme for tag and data requires the following elements:

- A parity bit for each tag.
- A parity bit for each data.
- An encoder for the incoming tag.
- A k_1+1 bit comparison logic for each way.
- A parity check circuit for the selected data.

Although the number of parity check bits is the same as for the SPP1 scheme, the logic circuitry to access the memory is simpler (as discussed in [16]).

III. PROPOSED PROTECTION SCHEME

As per the discussion in the previous section, the avoidance of false positives for tags is necessary because they are the only events that can lead to a failure (false negatives can only cause a miss). However, the scheme proposed in this paper relies on an additional yet subtle observation that false positives on the stored tags do not need to be detected or avoided prior to comparison. The important attribute is that the final result is not a hit and incorrect data is not provided as output of the associative memory. So, based on this observation, an alternative is to use a Combined Tag and Data Parity (CTDP) for each tag and its associated data. Then false positives will be detected when checking the combined parity and the data is read. On a parity error, the entry can be invalidated and a miss is generated. In the rest of this section, read-only and

Fig. 3. The proposed Tag and Data parity protected (CTDP) n -way cache.

write-through caches are used as a case study to illustrate the proposed CTDP scheme.

Figure 3 illustrates the proposed CTDP scheme used for caches. The combined parity P_c for each tag and associated data is stored with the data. Then on a read operation, the incoming tag is first compared with the tags stored in the cache position that corresponds to the incoming address. If there is a match, then the data and the combined parity are retrieved. Before returning the data, the stored parity is compared against the parity of data and the parity of the incoming tag. If there is an error in the parity, a miss is returned as result of the read operation (i.e. if there is one bit error in the stored data, the parity check will detect it).

Next, consider the occurrence of a single bit error in one of the stored tags and then, matching the incoming tag. The parity of the incoming tag and the read data will also not match the stored parity and thus, the false positive is detected. The significant difference with the traditional separate parity protection is that now the false positive is detected at the end of the read operation. This shows that the proposed CTDP scheme is also capable to protect against single bit errors rather efficiently.

As per Figure 3, the proposed CTDP scheme for the tags and data requires the following elements:

- A parity bit for each tag and its associated data.
- A parity calculation circuit that performs the same function as the encoder in Figure 2, but it only needs to output one bit (the calculated parity bit) for the incoming tag.
- A 1 bit comparison circuit, i.e. an *xor* gate.
- A parity check circuit for the selected data.

Next a comparison between the circuitry needed for the proposed CTDP scheme and the traditional separate parity protection is pursued. The differences between these two classes of schemes are highlighted in Figures 1, 2 and 3. In Figures 1 and 2, the elements in orange are those that are not needed for the CTDP scheme, the elements in yellow in Figures 2 and 3 are not needed for the SPP1 scheme. The only additional element for the CTDP scheme is the 1-bit

Fig. 4. Read operation on a cache protected by different schemes: SPP1 (left), SPP2 (middle) and CTDp (right).

comparison circuit shown in green in Figure 3. So, the CTDp scheme reduces the number of parity bits needed; moreover, the CTDp scheme only performs the parity check for the incoming tag (instead of computing it for the tag stored in each way). If a direct comparison is employed [16], the parity would also be computed only for the incoming tag. However, in such case, the parity bits stored in each way must be compared with the computed parity; therefore, the circuitry is still more complex than for the CTDp scheme. In terms of delay, the parity of the incoming tag in the CTDp scheme is computed in parallel with the access and the comparison with the stored tags, i.e. the parity computation is no longer in the critical path.

The read operation on caches for all three schemes is presented in the flow diagrams of Figure 4 (differences are also marked using the same colors of Figures 1 to 3). From the

figures and the above discussion, the CTDp scheme reduces the cost of protecting caches; this finding will be corroborated by the results presented in the next section. Note also that the proposed CTDp scheme can also be extended to scenarios in which stronger error detection codes are used for multiple errors as applicable to SPP1 and SPP2 too.

IV. EVALUATION

The evaluation of the proposed CTDp scheme is pursued in this section; it consists of three parts, which are analyzed in the next subsections:

- Four L1 Instruction caches with different configurations have been implemented and a comparison for the circuitry needed by the different protection schemes, excluding the memory, is given.

TABLE 1
COMPARISON FOR L1 INSTRUCTION CACHES IN DIFFERENT PROCESSORS FOR DIFFERENT PROTECTION SCHEMES

Cache configuration					Scheme	Area		Delay		Power	
Associativity	Tag	Data	Line-size			μm^2	%	ns	%	mW	%
LEON UT700 [19]	4-way	19-bit	32-bit	32 Bytes	SPP1	1406.0	100	0.95	100	0.87	100
					SPP2	1219.6	86.74	0.93	97.89	0.82	94.25
					CTDP	1190.4	84.67	0.77	81.05	0.79	90.80
RISC-V [20]	2-way	21-bit	64-bit	64 Bytes	SPP1	1157.2	100	0.80	100	0.78	100
					SPP2	1098.0	94.88	0.79	98.75	0.77	98.72
					CTDP	1076.8	93.05	0.65	81.25	0.75	96.15
Base Cache [9]	4-way	21-bit	64-bit	32 Bytes	SPP1	2362.0	100	1.00	100	1.61	100
					SPP2	2180.0	92.29	0.97	97.00	1.49	92.55
					CTDP	2128.4	90.11	0.83	83.00	1.47	91.30
ARM Cortex-R5 [21]	4-way	22-bit	64-bit	32 Bytes	SPP1	2395.6	100	1.01	100	1.63	100
					SPP2	2208.0	92.17	0.99	98.02	1.50	92.02
					CTDP	2151.2	89.80	0.85	84.16	1.48	90.80

- The implication of changing the width of the tag/data and the associativity has been studied.
- The number of additional memory cells in the cache array that need to be added for storing the parity bits has been evaluated for all different cases.

> Protection Circuitry

The proposed CTDP protection scheme has been studied for several L1 Instruction caches (ICs) configurations used in different processors; designs have been implemented in HDL and mapped to a 65-nm library from TSMC using Synopsys Design Compiler (with the default toggle rate). Two separate parity protection schemes (SPP1 and SPP2) have also been evaluated to show the benefits of the proposed scheme.

The cache design (L1) for the following processors have been considered in the evaluation:

- 32-bit LEON UT700 processor with 19-bit tags for the 4-way configuration [19].
- 64-bit RISC-V processor with 21-bit tags for the 2-way configuration [20].
- 64-bit Base Cache with 21-bit tags for the 4-way configuration [9].
- 64-bit ARM Cortex-R5 processor with 22-bit tags for the 4-way configuration [21].

The synthesis tool has been set to area and delay optimization in the parity protection circuits, excluding the memory, to obtain the best results for these metrics; the results are given in Table 1. The proposed CTDP scheme incurs in the lowest overheads for the protection circuitry in terms of area, delay and power consumption for all cases. For example, in the case of protecting the L1 in LEON UT700 processor, the CTDP scheme reduces 15.33% the area, 18.95% the delay, and 9.20% the power consumption. These results confirm that the CTDP circuit is simpler than existing schemes.

B. Cache Configurations

In addition to the four cache configurations reported in Table 1, the benefits of CTDP have been evaluated for different degrees of associativity and data widths.

For associativity, caches with 19-bit tags and 32-bit data have been evaluated in the range from 1-way to 16-ways; the synthesis results are reported for the area in Figure 5, for the delay in Figure 6, and for the power consumption in Figure 7. From the results, the reductions achieved by CTDP increase with associativity.

To assess the impact of data width, caches in a 4-way associativity with different data widths have also been evaluated. In particular, 11-bit tags and 16-bit data, 19-bit tags and 32-bit data (for LEON), 22-bit tags and 64-bit data (for ARM), and 26-bit tags and 128-bit data are also compared as examples. The synthesis results for these caches are shown for the area in Figure 8, for the delay in Figure 9, and for the power consumption in Figure 10. Again, CTDP provides reductions for all three parameters but in this case, there is no clear trend with data width.

From the results, it can be seen that regardless of the number of ways or the width of data, the proposed CTDP scheme offers a substantial advantage in terms of protection circuit overhead for all cases considered.

Fig. 5. Cache area vs. number of ways for different protection schemes.

Fig. 6. Delay vs. number of ways for different protection schemes.

Fig. 7. Power consumption vs. number of ways for different protection schemes

C. Memory Size for Protection

In this subsection, the overhead in memory size due to the cells to store the parity bits in the cache array is assessed and compared with previous schemes.

Define the width of each tag as k_1 and each stored data as k_2 . Then for a cache with E lines, M data words per line and a

Fig. 8. Cache area vs. width of Data for different protection schemes

Fig. 9. Delay vs. width of Data for different protection schemes.

Fig. 10. Power dissipation vs. width of Data for different protection schemes

n -way configuration that utilizes the proposed CTDp scheme, the total number of memory cells is given by $(k_1 + (k_2 + 1)M) \cdot n \cdot E$; $((k_1 + 1) + (k_2 + 1)M) \cdot n \cdot E$ cells are needed when utilizing the two separate parity protection schemes. As for the number of cells that store the parity bits, M cells are needed per line for the CTDp scheme and $M + 1$ for the SPP schemes. Therefore, the reduction ratio of the proposed CTDp scheme and the SPP1/SPP2 schemes in terms of memory cells that store the

parity bits per line is given by:

$$\text{Reduction ratio} = 1 - \frac{M}{M + 1} \quad (1)$$

Based on equation (1) for the caches given in Table 1, 11.11% of the memory cells that store the parity bits can be saved by the proposed scheme when protecting the 32-bit cache that has a line size of 32 bytes (i.e. 8 words per line) in the LEON UT700 processor and the cache in the RISC-V processor; 20% of the memory cells can be saved when protecting the base cache and the cache in the ARM Cortex-R5 processor. As memories in most cases account for a significant fraction of the circuit area of processors, this makes the proposed CTDp scheme very attractive for protecting caches in many applications.

V. EXTENSION FOR STRONGER CODES

As discussed previously, parity protection schemes are also applicable to scenarios in which interleaved error detection or stronger error correction codes are used when dealing with multiple errors. In this section, the extension of the proposed CTDp to stronger codes is further studied.

The proposed scheme is applicable to read-only or write-through caches (in which only false positives on tags and errors on data may lead to a failure); so error detection codes can be utilized for these caches. This leads to the use of interleaved SED codes in the presence of multiple bit errors because they can detect errors that affect adjacent bits on each word. In the implementation for this class of codes, t parity check bits are added to the unprotected information by encoding the t -distance interleaved bits to detect t -bit burst errors [22]. In this case, more memory cells in each cache line can be saved by using the proposed CTDp scheme because more parity bits are needed for tags and data in the separate parity protection schemes; so, the overhead reduction ratio is the same as that given previously in equation (1).

When error patterns are random and not necessarily located in adjacent positions, stronger codes that can detect and correct multiple errors should be utilized. A widely used class of ECCs is Single Error Correction-Double Error Detection (SEC-DED) codes; these codes are derived from the SEC Hamming codes and use an extra parity bit to extend the Hamming distance from three to four, so that double bit errors can be detected. For Hamming codes, the relationship between the number of parity bits r and the unprotected information size k is given by:

$$k = 2^r - r - 1 \quad (2)$$

where k and r are positive integers and $r \geq 3$. So, $r + 1$ parity bits are required for the SEC-DED codes. If the width of the information to be protected is not the root of equation (2), then SEC-DED codes can also be shortened to fit the information size by using the same number of parity bits. In this case, equation (2) is now given by:

$$k' = 2^r - r - 1 - l \quad (3)$$

where k' is the unprotected information size, and l is the difference between k' and k that needs to be shortened and $1 \leq l \leq k$.

> Based on equation (2) or (3), when using SEC-DED codes to protect caches, the number of memory cells that store the parity bits in the tag RAM is usually smaller than in the data RAM because tags are shorter than data. Define r_1 as the number of parity bits for each tag and r_2 for each data, then the reduction ratio given previously by equation (1) is now given by:

$$\text{Reduction ratio}' = 1 - \frac{M \cdot r_2}{r_1 + M \cdot r_2}. \quad (4)$$

By comparing equation (1) with (4), the relationship of the Reduction ratio when changing the interleaved SED to the SEC-DED codes can be established by:

$$\frac{\text{Reduction ratio}'}{\text{Reduction ratio}} = \frac{r_1 + M \cdot r_1}{r_1 + M \cdot r_2}. \quad (5)$$

There are two scenarios to consider:

- The memory overhead reduction ratio is the same if $r_1=r_2$;
- The memory overhead reduction ratio becomes worse if $r_1 \neq r_2$.

The first scenario occurs when the protected cache has the same value of k or k' in equation (2) or (3) so fitting the widths of tags and data. The second scenario, in which the ratio is worse, is most likely in practical configurations. This is because the difference between the widths of tags and data always makes them fitting different block sizes of codes. For example, in the case of protecting the 32-bit cache with 19-bit tags in the LEON UT700 processor, $l=7$ and $r=6$ in equation (3) for the case of tag size $k'=19$; while $l=25$ and $r=7$ for the case of data size $k'=32$. So, seven parity bits (i.e., $r_1=7$) are needed for each tag, and eight (i.e., $r_2=8$) for each data; the memory overhead reduction ratio will decrease from 11.11% to 9.86% (as per equations (1) and (4)).

The proposed CTD protection scheme can also be extended to stronger ECCs, such as the Double Error Correction Bose Chaudhuri Hocquenghem (DEC-BCH) codes used in the SPP2 scheme of [17]. However, as shown in a previous section, error detection codes can provide sufficient protection for read-only or write-through caches. In this case, error correction codes would introduce an over protection as false negatives in those caches would not affect the correctness of the results; the number of required memory cells that store the parity bits would increase linearly with an increase of line-size. Also the complexity of the protection circuitry would increase in the same fashion, i.e. linearly.

Next, three cases are considered as examples to evaluate the overhead introduced by over protection of the ECCs:

- single bit errors;
- double adjacent bit errors;
- double bit errors (both adjacent and non-adjacent).

The number of memory cells needed to store the parity bits required by the SED codes and the SEC-Hamming codes are compared in the first case, the 2-bit interleaved SED (2I-SED) codes, the SEC-DED codes and the DEC-BCH codes are compared in the second case, and the SEC-DED codes and the DEC-BCH codes are compared in the last case. The numbers of parity bits that are required by different codes for various widths of information are given in Table 2 [22]. Then for the

TABLE 2
NUMBER OF PARITY BITS NEEDED FOR DIFFERENT CODES

Code	# of information bits k	# of parity bits r
SED	19	1
	21	1
	22	1
	32	1
	64	1
SEC-Hamming	26	5
	57	6
	120	7
2I-SED	19	2
	21	2
	22	2
	32	2
	64	2
SEC-DED	26	6
	57	7
	120	8
DEC-BCH	21	10
	51	12
	113	14

TABLE 3
COMPARISON FOR CACHE OVERHEAD DUE TO DIFFERENT CODING SCHEMES

Scheme		Cache overhead			
		LEON UT700 [19]	RISC-V [20]	Base Cache [9]	ARM Cortex-R5 [21]
SE	SED	X	X	X	X
	SEC	6X	7X	7X	7X
DAE	2I-SED	X	X	X	X
	SEC-DED	3.5X	4X	4X	4X
	DEC-BCH	6X	7X	7X	7X
DNAE	SEC-DED	X	X	X	X
	DEC-BCH	1.71X	1.75X	1.75X	1.75X

SE denotes single error, DAE denotes double adjacent errors and DNAE denotes Double Non-Adjacent Errors.

four caches given in Table 1, the total number of memory cells to store the parity bits in each line when utilizing different protection schemes are reported in Table 3. In the presence of single bit errors, the SEC-Hamming codes introduce 500% more parity memory cells than the SED codes to each line of the cache of the LEON UT700 processor (with 32-bit data); in the presence of double adjacent bit errors, the SEC-DED codes and the DEC-BCH codes introduce 250% and 500% more cells than the 2I-SED codes; finally in the presence of double non-adjacent bit errors, the DEC-BCH codes introduce 71.4% more cells than the SEC-DED codes.

VI. DISCUSSION AND CONCLUSION

Although focused on the protection against single bit errors, the principles presented in this paper can be extended to more powerful codes, such as interleaved parities that are effective for detection of multiple adjacent errors in conventional memories [23], or limited magnitude errors in emerging memories [8]. In these cases, the saving in memory will be more pronounced as more parity bits are needed per entry.

The application of the proposed CTD protection scheme to Content-Addressable memories (CAMs) used in high-speed

networking applications [24] is also of interest. CAMs are used to perform fast lookup against a set of keys in a number of applications. For example in Ethernet switching [25]), a CAM returns the address of the matching key that determines an associated output port. In this application, only a false positive can generate an error (a packet is sent to an incorrect port), while a false negative is not an issue (the switch will then launch an address resolution request to fill in the contents of the CAM). Therefore, the proposed CTDP scheme can be used to efficiently protect CAMs against soft errors. In this case, CTDP introduces an additional advantage in terms of overhead reduction because a CAM cell (using either CMOS or emerging techniques [26]) is typically more complex than a standard RAM cell.

The proposed scheme is also applicable to write-back caches. In this case, both the tags and data can be protected by a single code; on a match, the code is used to detect and correct errors. Assume that a Single Error Correction (SEC) code is used; then if there is a single bit error, it can be determined if it has affected a tag or data. If it has affected a data, the match is valid and the erroneous bit on the data must be corrected. If it has affected a tag, the match is not valid so the cache should not produce a match; instead it should fix the error on the tag. When an error on tag creates a false negative, a match could not be generated. To avoid this scenario, a check of the combined codes for the tags and data should be performed on every cache miss. This will add penalties (retrieve data for all ways and compute the codes), but provided the miss rate is low, in most cases it is acceptable. A detailed design for applying the CTDP scheme to write-back caches is left as future work. Moreover, for a hybrid memory system, the overhead reductions achieved by CTDP enable a faster operation of the associative memories, as well as a reduction of area and power, thus improving overall system performance.

Therefore, this paper has presented the Combined Tag and Data Parity (CTDP) scheme to reduce the cost of protecting associative memories against false positives due to soft errors. Based on the observation that only false positives can lead to a failure in some memories, the CTDP scheme detects a false positive when reading the value at the end of the read operations. Instead of using separate parities this enables the use of a single parity bit for the tag and data in an entry. The proposed CTDP has been studied for caches (read-only and write-through); it has been evaluated for different caches and the results show that it provides reductions in both cache overhead (in terms of additional memory cells used to store the parity bits) and in the protection circuitry (in terms of area, delay and power consumption). As an example, when protecting the L1 Instruction cache of a LEON UT 700 processor, the proposed scheme achieves reductions of 11.11% in the memory cells that store the parity bits, and 15.33% in area, 18.95% in delay, and 9.20% in power consumption for the protection circuitry.

REFERENCES

- [1] T. Takano, T. Yamada, K. Shutoh, et al., "In-Orbit Experiment on the Fault-Tolerant Space Computer Aboard the Satellite Hiten", *IEEE Trans. on Reliability*, vol. 45, no. 4, pp.624-631, 1996.
- [2] Y. Nezzari, C.P. Bridges, "ACEDR: Automatic Compiler Error Detection and Recovery for COTS CUP and Caches", *IEEE Trans. on Reliability*, vol. 68, no. 3, pp.859-871, 2019.
- [3] A. Amirany, F. Marvi, K. Jafari, et al., "Nonvolatile Spin-Based Radiation Hardened Retention Latch and Flip-Flop", *IEEE Transactions on Nanotechnology*, vol. 18, pp. 1089-1096, 2019.
- [4] S. Wang, J. Hu, and S. G. Ziaavras. "Replicating Tag Entries for Reliability Enhancement in Cache Tag Arrays." *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 4, pp. 643-654, 2012
- [5] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A state-of-the-Art Review," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 124-134, 1984.
- [6] A. Neale and M. Sachdev, "A New SEC-DED Error Correction Code Subclass for Adjacent MBU Tolerance in Embedded Memory", *IEEE Trans. on Device and Materials Reliability*, vol. 13, no. 1, pp. 223-230, 2013.
- [7] A. Das, N. A. Touba, "Efficient One-Step Decodable Limited Magnitude Error Correcting Codes for Multilevel Cell Main Memories", *IEEE Transactions on Nanotechnology*, vol. 18, pp. 575-583, 2019.
- [8] S. Liu, P. Reviriego, F. Lombardi, "Detection of Limited Magnitude Errors in Emerging Multilevel Cell Memories by One-Bit Parity (OBP) or Two-Bit Parity (TBP)", *IEEE Transactions on Emerging Topics in Computing*, 2019 (early access).
- [9] C. Zhang, F. Vahid, W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems", *IEEE 30th Annual International Symposium on Computer Architecture (ISCA)*, pp.136-146, 2003.
- [10] R. Van Der Pas, "Memory Hierarchy in Cache-based Systems", Sun Blueprints, pp.1-26, 2002.
- [11] M. Lapedus, "The Next New Memories", Semiconductor Engineering, [online] <https://semiengineering.com/the-next-new-memories/>, 2019.
- [12] G. Sun, J. Zhao, M. Poremba, et al., "Memory that Never Forgets: Emerging Nonvolatile Memory and the Implication for Architecture Design", *National Science Review*, vo. 5, no. 4, pp. 577-592, 2018.
- [13] Y. Xie, "Modeling, Architecture, and Applications for Emerging Memory Technologies", *IEEE Design & Test of Computers*, vol. 28, no. 1, pp. 44-51, 2011.
- [14] T. Wojcicki, *VLSI: Circuits for Emerging Applications*, CRC Press, 2017.
- [15] S. Mukherjee, "Architecture Design for Soft Errors", Morgan Kaufmann, 2011.
- [16] W. Wu, D. Somasekhar and S. L. Lu, "Direct Compare of Information Coded with Error-Correcting Codes", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 2147-2151, 2012.
- [17] S. Liu, P. Reviriego and L. Xiao, "Evaluating Direct Compare for Double Error Correction Codes", *IEEE Trans. on Device and Materials Reliability*, vol.17, no.4, pp. 802-804, 2017.
- [18] A. Gendler, A. Bramnik, A. Szapiro, et al., "Don't Correct the Tags in a Cache, Just Check Their Hamming Distance from the Lookup Tag", *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp.571-582, 2018.
- [19] "UT699E/UT700 LEON 3FT SPARC™ V8 Microprocessor Functional Manual", Version 1.2.4, <https://www.cobham.com/HiRel>, 2018.
- [20] M. Noltner-Augustin, "RISC-V Architecture and Interfaces the RocketChip", Advanced Seminar "Computer Engineering", University of Heidelberg WT16/17, pp.1-6, 2016.
- [21] "Cortex-R5 and Cortex-R5F Technical Reference Manual", Revision:r1p1, <https://developer.arm.com>, 2011.
- [22] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.
- [23] M. Maniatakis, M. K. Michael, Y. Makris, "Multiple-Bit Upset Protection in Microprocessor Memory Arrays Using Vulnerability-Based Parity Optimization and Interleaving", *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.23, no.11, pp.2447-2460, 2015.
- [24] K. Pagiamtzis and A. Shekholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey", *IEEE Journal of Solid-State Circuits*, vol. 41, no.3, pp.712-727, 2006.
- [25] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A Soft-Error Tolerant Content Addressable Memory (CAM) using an Error-Correcting-Match Scheme," *IEEE Custom Integrated Circuits Conference*, pp. 301-304, 2006.
- [26] P. Junsangri, J. Han, F. Lombardi, "Design and Comparative Evaluation of a PCM-Based CAM (Content Addressable Memory) Cell", *IEEE Transactions on Nanotechnology*, vol. 16, no. 2, pp. 359-363, 2017.