Implicit-Storing and Redundant-Encoding-of-Attribute Information in Error-Correction-Codes

Yiannakis Sazeides
University of Cyprus

Emre Özer
ARM

Danny Kershaw
NXP

Panagiota Nikolaou
University of Cyprus

Marios Kleanthous
University of Cyprus

Jaume Abella
Barcelona Supercomputing Center

ABSTRACT

This paper proposes implicit-storing to extend the logical capacity of a memory array without increasing its physical capacity by leveraging the array's error-correction-codes to infer the implicitly stored bits. Implicit-storing is related to error-code-tagging, a technique that distinguishes between faults in data and invariant attributes of a location when the attributes are not stored in the memory array but are encoded in the error-correction-codes. Both error-code-tagging and implicit-storing cause a code-strength reduction due to their encoding of additional information in the code meant to only protect data.

Redundant-encoding-of-attributes is introduced to improve the strength of a code by encoding same information in multiple codewords in a cache or memory. We demonstrate how EREA and IREA, two derivatives of redundant-encoding, alleviate the code-strength reduction experienced by error-code-tagging and implicit-storing respectively.

Implementing the proposed methods requires minor modifications in the encoding and decoding logic of the baseline error-correction scheme used in this work. The paper discusses several uses of the proposed schemes to help demonstrate their usefulness.

Categories and Subject Descriptors

B.4.5 [**Reliability, Testing, and Fault-Tolerance**]: Hardware reliability; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; C.4 [**Performance of Systems**]: Fault tolerance

General Terms

Design, Reliability

Keywords

Implicit Storing , Error Code Tagging, Redundant Encoding, Memory, Reliability, Error Correction Codes

1. INTRODUCTION

Error-correction-codes (ECC) [11] are commonly employed to protect data in caches and main memory from faults due to particle strikes [34], hard or repeated failures [26, 14], operation close or even below V_{min} [32], static [6] and dynamic [7] variations etc. The importance of ECC has been increasing with smaller feature size due to the exponential growth of memory elements integrated in processor and memory chips.

When data is written to an ECC protected memory array check bits derived from the data are also stored along with the data in the array. The ECC check bits are redundant information that encodes the data and are used on a read to detect an error and determine whether it is possible to correct it and how.

An ECC code is usually described by the number of check bits it uses, k, the number of data bits it protects, m, and its strength, how many bit errors it can detect and correct. Very often, due to storage organization requirements, the ECC code is shortened [25]: the number of protected data bits, m, is smaller than the maximum number that can be protected by k check bits of an ECC code. Consequently, the k check bits have the potential to provide protection for additional data bits that remain underutilized.

One possible use of the extra coding capacity is to encode invariant attributes that are associated with the data in a location (e.g. the memory block address). These attributes do not need to be stored in the memory array if they are available each time the data are accessed. This enables to check that the read data are both correct and with the right attribute encoding. We refer to this approach as error-code-tagging (ECT), after a related proposal by Gumpertz [10].

First, we propose to use the extra coding space for erasure coding [9] of some of the attribute bits. Erasure coding is a well-known technique for correcting errors when the position of an error is known, and is widely used for disk error protection [24]. We propose to implicitly-store attribute bits by "intentionally" erasing some of the attribute bits and infer them on reads. We referred to this as implicit-storing (IS). The proposed approach enables us to extend the logical capacity of a memory array protected with shortened ECC code without increasing its physical capacity, i.e. storage for free.

The use of ECT or IS leads to an ECC code strength reduction because part of the ECC code intended to detect and correct errors is used in the case of ECT to check for attribute errors and in the case of IS to store and infer the erased bits. For example, as explained in Section 4.2, by implicitly-storing one bit with a single-error-correction double-error-detection code we lose the ability to detect some double data errors. Although this may be an acceptable

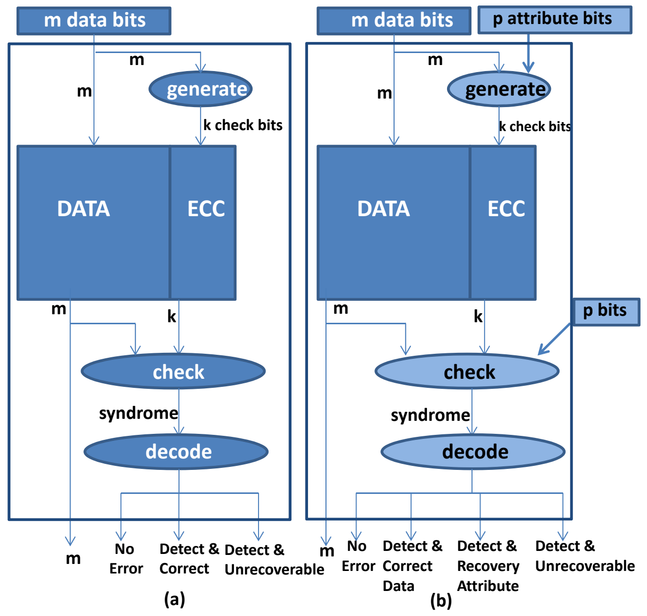


Figure 1: (a) ECC Protection (b) ECT Protection

trade-off, (e.g. reducing area or energy overhead for lower fault-coverage), it is desirable to minimize the code strength reduction as much as possible.

To help mitigate the code-strength reduction of ECT or IS we exploit a common property of caches and memory [15, 3, 19, 22, 5]: the granularity used for ECC code protection, e.g. 64-bit word, is often smaller than the block size, e.g. 512 bits block. In particular, we propose to redundantly encode in two or more codewords in each block the same ECT attributes or implicitly-stored information. We refer to this approach as Redundant-Encoding-of-Attributes (REA). Fundamentally, REA is similar to n-way modular redundancy [18] and helps to improve fault tolerance.

ECT and IS can be applied independently of REA but this work considers their combined use: ECT+REA referred to as Explicit Redundant Encoding of Attribute Information (EREA), and IS+REA referred to as Implicit Redundant Encoding of Attribute Information (IREA). The paper shows how EREA can improve the fault-coverage of ECT, and, similarly, how IREA recovers some of the code-strength reduction of IS. All proposed methods require minimal design complexity and have several uses that help improve fault tolerance, energy efficiency, security and performance.

The rest of the paper is organized as follows: Section 2 covers background related to ECC, erasure codes and error-code-tagging. Section 3 describes the proposed implicit-storing technique. This is followed by the description of REA, EREA and IREA in Section 4. Various use-cases of the proposed schemes are discussed in Section 5. Section 6 presents an area, delay, power and scalability analysis. The related work is discussed in Section 7, and the paper concludes in Section 8.

2. BACKGROUND

This section provides background on memory array protection, presents the specific ECC code assumed in the paper, reviews error and erasure correction, and describes error-code-tagging.

2.1 ECC Protected Data Array

Fig. 1(a) presents the generic organization of an ECC protected array. An array consists of many blocks, b, each containing a fixed number of data bits, s. For the purposes of ECC protection the block is divided into w equal size words (each with $m{=}s/w$ data bits). The array stores k check bits for each of the m-bit data words. Every time a word is written, the generate unit produces its k-bit check bits, according to the ECC used, that are stored in the array together with the data. Every time a word is read from the array the check unit takes as input the word's data and check bits and produces a syndrome. The syndrome is decoded to determine whether or not an error has been detected. When an error is detected, the error is corrected if it is a correctable one. Otherwise, an unrecoverable error is flagged.

The top side of Fig. 1(a) (i.e. generate) depicts what happens on a write operation, and the bottom side of the figure (i.e. check and decode) shows what happens on a read operation. The same convention is adopted in the other figures of the paper.

2.2 SEC-DED ECC Code

A category of linear error detection and correction codes that can correct single and detect double errors [11], known as SEC-DED codes, are very popular due to their low overhead and high fault coverage. Hsiao code [12] is often cited as the most cost-effective SEC-DED scheme due to the minimum number of XOR gates it uses to generate its check bits. In particular, a Hsiao code, henceforth also referred to as code, needs an n bit codeword to protect m bits of data, where $n{>}m$, the number of check bits $k{=}n{-}m$, and $m{\leq}\ 2^{k-1}{-}k$.

A Hsiao code can be uniquely defined by its check (or generator) matrix. A check matrix, H, consists of k rows and n columns. Hsiao code requires a check matrix that contains unique columns with odd number of 1s. Each row is an n bit vector that defines which data and check bits are used for computing a syndrome bit. A syndrome is a k-*bit* vector that is used to check the integrity of an n-*bit* information word. It is obtained by performing the product of the kxn check matrix, H, with the nx*1* codeword. The syndrome is decoded as follows:

1. equals to the zero vector: no error is detected,
2. contains an odd number of 1s and is equal to one of the check matrix columns: a single bit error is detected and its position corresponds to the column position in the H matrix,
3. contains odd 1s but does not exist in the H matrix: It will be treated as unrecoverable error, and
4. has an even (greater than 0) number of 1s: it is decoded as an unrecoverable error (for example any two bit faults will result in an even syndrome, however, some rare combinations of even errors ≥ 4 may result to a zero syndrome and remain undetected).

Fig. 2 describes the possible actions with different number of actual errors in a codeword and also indicates whether the action is correct. The behavior is as expected with 0, 1 and 2 errors. When the number of errors exceeds the strength of the code we may have incorrect decisions. For example, when there are three errors in some cases they are detected as unrecoverable, but in the remaining cases they are incorrectly treated as single bit errors that get miss-corrected (a correction that does not recover the correct data).

	Number of Actual Errors				
	0	1	2	3	
SEC-DED Decoder	No error	Detect odd & Correctable	Detect even & Unrecoverable	Detect odd & Unrecoverable	Detect odd & Correctable
Correct Decision	YES	YES	YES	YES	NO (miss-correction)

Figure 2: SEC-DED behavior with different number of actual errors

Unless indicated otherwise the rest of the presentation assumes the code discussed in this Section. Nonetheless, we like to note that the techniques proposed in the paper are applicable to other linear codes (SEC-DED or stronger).

2.3 Hamming Distance, Errors and Erasures

The strength of a code is a function of its minimum Hamming distance, D_{min}, between any of its legal codewords [11]. In general, a code has the strength to detect all errors up to d in any codeword where $d \leq D_{min}$-1, or can correct all errors up to e where $e < \lfloor D_{min}/2 \rfloor$ [11]. Furthermore, a code can correct r erasures where r=D_{min}-1. An erasure is a specific bit position of a codeword with an unknown value [9], this is different from an error that corresponds to a bit flip in an arbitrary bit position in a codeword. A given code can correct more erasures than errors because it is easier to correct when the problematic positions are known.

We illustrate with an example the difference between errors and erasures in Fig. 3 assuming a 7-4 SEC-SED (single-error-correction and single-error-detection) code and a 7-4 double-erasure code. The example assumes a 4-bit zero value. The encoder is the same in both cases and generates the error correction code. The positions 1, 2, 3 and 4 represent the data bits and the remaining positions, 5, 6 and 7, represent the parity bits. For erasure code the erased positions are 1 and 3. Let's assume a double bit error in positions 1 and 3. Fig. 3(a) shows how the erasure code recovers the erased bits by considering all their value combinations and selecting the combination that results in an ECC match. However, in the case of double errors in other positions this scheme will not work. On the other hand, in Fig. 3(b) a double bit error exceeds the SEC-SED code strength and the decoder miscorrects the data.

The number of error detections, corrections and erasures offered by a specific decoding of a code depends on the D_{min} of the code and the choice of d, e, r [25]. For example, a code at the same time can detect d and correct e errors as long as e+d<D_{min} or at the same correct e errors and r erasures given that $2e$+r<D_{min}. Specifically, a code with D_{min}=4 can be used for three error detections, SEC-DED, three erasure corrections, or one error correction and one erasure correction, one erasure correction and double error detections or two erasure corrections and one error detection.

Typically, processor caches and main memory rely on codes that detect and correct errors since the positions of the errors are not known a priori. On the other hand erasure codes are used to recover lost data in redundant arrays of disks since the specific position in the data corresponding to a failing disk is known [24]. Nonetheless, schemes inspired by erasure coding have recently been proposed for cache protection where one code is used to detect the error position, and a second code is used for erasure correction [16, 20].

In this paper, we show how to augment a code used for memory array error correction and detection to also perform erasure correction without storage area overhead and minimal code strength reduction.

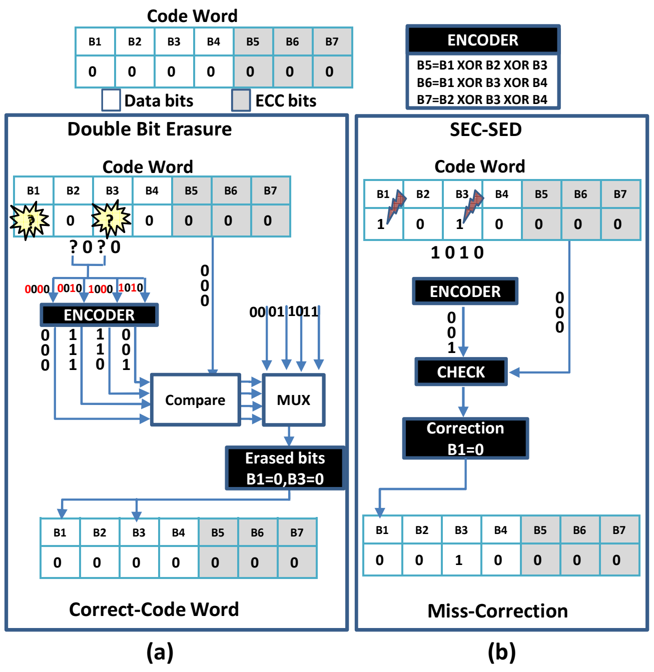


Figure 3: Example with (a) Double bit Erasure and (b) Double bit Error

2.4 Error-Code-Tagging

Very often due to storage configuration an ECC code, such as the one presented in Section 2.2, is shortened. That is the number of protected data bits, m, is smaller than the maximum number that can be protected by k check bits. Consequently, the k check bits can provide protection for up to p=2^{k-1}-k-m additional data bits. For example, when data is 64 bits, m=64, the number of check bits, k, must be at least 8, and, therefore the code can protect up to 2^{8-1}-8=120 bits. However, since the data size is m=64 bits, there is room for additional 56 bits of data that can be protected with the very same 8 check bits. These are 56 bits that can be used to implement error-code-tagging (ECT) [10] by encoding additional attributes that are associated with the data. ECT does not store the attributes in the storage and requires the attributes to be available each time the data are accessed.

Fig. 1(b) shows how an ECC scheme can be extended with ECT capabilities. The most notable addition, as compared to Fig. 1(a), is the extra inputs (p attribute bits) used to generate the check bits and the syndrome on write and read respectively. Note that these attributes bits are not saved in the storage because they are available whenever the data is accessed and remain invariant between two writes, i.e. a read from a location is guaranteed to receive the same attribute as the one used on the most recent write to the same location.

To illustrate the ECT concept, consider the case of a cache sub-system in a processor core where each data access is accompanied by an attribute, e.g. the line address or security ID. When the data is written into the cache both the data and attribute are used to generate the ECC code that is stored in the cache. At the time of the read, when the core

	Data Errors			
Attribute Errors	**0**	**1**	**2**	
0	No error	Detect Odd Data & Correct	Detect Even & Unrecoverable **(cannot distinguish)**	
1	Detect Odd Attr. & Recovery	Detect Even & Unrecoverable **(cannot distinguish)**	Detect Odd & Unrecoverable	Detect Odd Data & Correct **(miss-correction)**

Figure 4: ECT behavior for different number of data and attribute errors

wants to access the same cache line, it provides the cache line address as an attribute. The ECC check logic takes as input the data and the ECC code read from the cache as well as the attribute provided by the core and checks for errors. If the data and the attribute given at the read time is the same as the data and attribute provided at the write, the ECC logic will give "No Error" status. However, if the data from a location is the same but the attribute, line address in this case, is different and the code detects it, then the ECC check logic will flag this as "Error". This error would have remained undetected if the ECC had encoded only the data of a location.

The main limitation of the ECT approach is that by extending protection to attributes an ECC code becomes unable to differentiate between the bit positions of multiple faults. For instance, a SEC-DED code extended with ECT is unable to differentiate in the case of two faults whether they are only in the data, only in the attributes, or one fault in both. Furthermore, an ECT scheme can miss-correct some error combination that will be detected as unrecoverable if attributes are not protected. Although these weaknesses of ECT may be worth disregarding for the extended protection ECT offers to attributes, it may be desirable to differentiate between these different fault events. For example, in a security scenario where the attribute is used as a security ID, it is better for any error detected in the security ID to lead to a system freeze or shutdown. Alternatively, if the attributes are used to encode an address, a retry can be initiated from a checkpoint (if one is available) to overcome a possible soft-error in address decoding/tag match logic. We show next in the paper how the use of redundant-encoding-of-attributes can help mitigate this weakness of ECT.

Fig. 4 shows the behavior of the ECT scheme for different combination of data and attribute errors assuming a one-bit attribute. The example behavior illustrated in Fig. 4 assumes a Hsiao SEC-DED code extended for ECT with the attribute assigned a distinct odd column in the H-matrix[1]. The table helps illustrate the benefits and weaknesses of ECT. It can detect single data or attribute errors precisely but (i) when an even number of data errors occur, it cannot distinguish between data and attribute faults (2 data, 1 data+ 1 attribute), and (ii) in some cases with 2 data and 1 attribute errors it can perform a miss-correction.

The rest of the paper assumes that a cache is protected with a shortened ECC code which is very common in practice [15, 3, 19, 22, 5]. We show next how to use the extra coding space of a shortened code to increase the effective cache capacity and to improve the strength of a code.

[1]Assigning more than one unused column of the SEC-DED code to attribute bits is possible but should be avoided if attribute errors are expected to be multi-bit ($\geq$2) and possible to occur at the same time with data error(s). Such error events can lead to miscorrection, misrecovery or even undetected errors. If an attribute has two or more bits it can either be hashed to fewer bits [10] or be split and assigned to different words in a block with each word checking a subset of the attribute.

3. IMPLICIT-STORING (IS)

The logical capacity of a cache protected with a shortened SEC-DED code can be increased by p bits per word without increasing its physical storage. This is accomplished by augmenting the shortened SEC-DED code to also perform a *p-bit* erasure correction. This logical capacity increase comes at the expense of reducing the ECC code-strength. We will postpone the discussion of what attributes can be implicitly stored and their use-cases until Section 5 and first examine the basic operation of implicit-storing.

3.1 Basic Operation of Implicit-Storing

On a write of an m bit data, the proposed mechanism stores m bits of data but computes the check-bits using the values of the p additional attribute bits. These p bits are intentionally erased and are said to be implicitly stored. The positions of the p implicitly stored bits in the codeword are known both to the encoder and decoder. Therefore, IS requires that the original shortened ECC code has at least p bit positions unused.

When the m bit data and k check-bits are read the decoder attempts to infer the missing p data bits by decoding 2^p times, each with a different value combination of p bits. The decoder considers the combination of the resulting 2^p syndromes to determine what the missing p-bit value is, and whether an error is detected in the codeword and if it can be corrected.

An important property of an ECC code is the maximum number of bits, p, which can erase. These correspond to the number of bits that can be implicitly-stored and inferred always correctly when there is no error in the m stored bits. For SEC-DED based codes [11, 12] this is equal to 3 bits, one less than the minimum hamming distance of the code (see Section 2.3 for a code's erasure strength). Although it is feasible to implicitly-store three bits using a SEC-DED code this takes up all the code-strength. We discuss in Section 6 the trade-off between code-strength and the number of implicitly-stored bits.

A generic description of the proposed mechanism is shown in Fig. 5. The key differences from the conventional ECC and ECT protection (see Fig. 1(a) and Fig. 1(b)) are: (i) the ability to logically store additional p-bits and infer them at read time (unlike ECT that provides the information at read time to check for errors in the attributes), and (ii) increased decoding overhead, illustrated qualitatively with the multiple checkers. In an actual implementation the multiple checkers may share logic or one checker may be used to iterate sequentially over the possible values of p. In fact, we show subsequently that for a SEC-DED code the logic overhead required by IS checkers is only few inverters.

3.2 Implementation of 1-bit Implicit-Storing

Here we present a specific implementation of IS for a Hsiao SEC-DED code. The code uses an n bit codeword to protect m bits of data, the number of check bits k=n-m, and

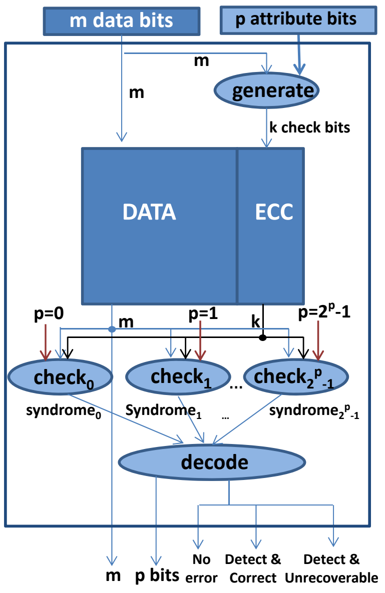


Figure 5: Array with ECC Protection and Implicit-Storing

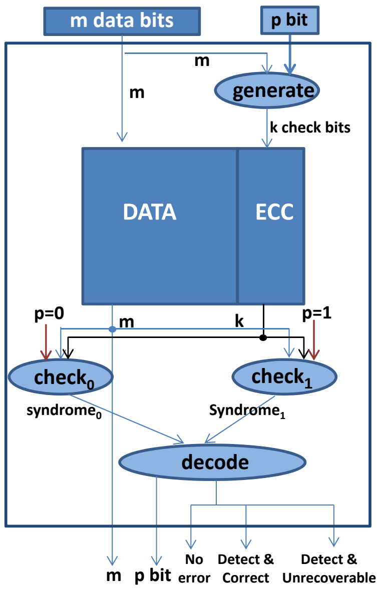


Figure 6: Array with SEC-DED Protection and Implicit-Storing of 1-bit

$m \le 2^{k-1}$-k. Let us assume a baseline with n=72, m=64 and k=72-64=8. That is the baseline code that requires 72 bits per storage word, 64 for the data and 8 bit for the check bits.

Fig. 6 shows how to implement the proposed scheme when the number of data bits implicitly-stored is one ($p = 1$, i.e. the logical storage size is increased by 1 bit per word). A comparison of Fig. 1(b) and Fig. 6 reveals that the two schemes are virtually the same except that IS uses two checkers instead of one. The generate unit of IS takes the erased bit as additional input to determine the check bits. The two checkers of IS have exactly the same inputs, all bits read from the array, except that one checker assumes the implicit value to be a 0 and the other 1. The syndrome decoder takes two syndromes as input, instead of one, and produces its output.

Fig. 7 describes the behavior of the proposed scheme for different number of actual errors. When there are no errors, the syndrome decoder infers correctly the missing bit. This corresponds to the implicit value assumed by the checker that indicates no error since the other checker detected a single correctable error due to the incorrect value for the implicit bit. When there is a single error the proposed scheme can detect and correct the error using the checker that indicates one detected error whereas the other checker detects two errors, the actual error and the one due to the wrong value of the implicit bit. When there are two errors there is a chance for miss-correction. In particular, the checker that assumes the correct implicit value will detect double error whereas the other will behave as if there are three errors. As shown in Fig. 7 three errors may lead to either detecting a 3-bit unrecoverable error or single-bit error. When the latter occurs we have a miss-correction. We have performed a miscorrection analysis, for the 72-64 SEC-DED code in [12], assuming uniformly distributed 64 bit values while using an available column with five 1s in the parity matrix [12] to represent the implicit bit. The minimum number of 1s in available columns of the matrix protecting 64 data bit is 5 [12]. We found out that in the presence of 2 data errors the expected probability for a miss-correction is 28.27% .

The last row of Fig. 7 shows the corresponding output of the syndrome decoder for the baseline SEC-DED. The behavior of IS and SEC-DED is the same for 0 and 1 errors. However, with 2 errors the proposed scheme, as noted earlier, can lead in some cases to a miss-correction.

The cost of IS is minimal. The generate unit is the same as the baseline ECC except the need to xor its output with the implicit value only in the positions that have 1s in the parity matrix column of the implicit bit. Since this column has 5 ones then five extra XOR gates will be needed.

The two check units of IS are equivalent to just one baseline check unit plus 5 inverters. A baseline check unit produces the syndrome0 that corresponds to the checker that assumes the implicit bit value is 0 since an implicit value of 0 does not change the syndrome value. To produce the syndrome1, that assumes the implicit bit value is 1, we simply invert five syndrome0 bit positions. The bits of syndrome0 that are not inverted are common to both syndrome0 and syndrome1. Fig. 8 shows the detailed implementation of IS with 1-bit attribute.

In summary, we can increase the capacity by one bit in each word of a cache array at the cost of (i) 5 XOR and 5 inverters and slightly larger decoder, and (ii) code-strength reduction in some cases of 2 data errors. In the next section, we present a technique that helps lessen the code-strength reduction due to implicit-storing.

4. REDUNDANT-ENCODING-OF-ATTRIBUTES (REA)

One of the basic ideas of this work is to redundantly encode attributes (REA) in multiple codewords to recover some of the code strength lost due to the encoding of additional attributes in a shortened ECC code.

REA exploits a common characteristic of caches and main memory: the granularity used for ECC protection, e.g. 64-bit word, is often smaller than the granularity of transfer, e.g. 512 bits block. We propose to encode in multiple words

Missing bit value is X	Number of Actual Errors			
	0	1	2	
Syndrome from checker that assumes missing Bit is X	No error	Detect odd & Correctable	Detect even & Unrecoverable	Detect even & Unrecoverable
Syndrome from checker that assumes missing bit is X'	Detect odd & Correctable	Detect even & Unrecoverable	Detect odd & Unrecoverable	Detect odd & Correctable
Decoder Decision	No error	Detect odd & Correctable	Detect even & Unrecoverable	Detect odd & Correctable
Infer Missing Bit	X	X	NA	X'
Correct Decision	YES	YES	YES	NO (miss-correction)
Infer Correctly	YES	YES	NA	NO
Baseline ECC	No error	Detect odd & Correctable	Detect even & Unrecoverable	

Figure 7: Behavior of IS with different number of actual errors and Implicit-Storing of 1-bit

of a block, e.g. two neighboring words, the same attributes. We refer to each of these group of words in a block as correlated.

Although REA is not applicable to memory arrays with block-wide ECC, apparently, many ECC designs in caches and memory systems use sub-block-based ECC, which will make REA applicable in practical cache and memory system implementations [15, 3, 19, 22, 5]. There are many practical reasons for this:

1. Design constraints (area, energy etc) or standards limit read/write granularity. For example, a 64B block may be read/written in eight 8B chunks (DRAM and caches).

2. If a higher level cache is write-through you can update the lower level cache at the granularity of word without having to read the whole block first.

3. The array area would be smaller with a block ECC vs word ECC. But, the delay (latency) would be larger. The time to decode a Hamming like ECC increases with the number of bits N. In theory, the additional decoding only grows with $\log_2 N$ - but in practice, as the logic gets bigger, the logic gets spread out and the wire delays can become large.

4. Avoid having to read the entire block before checking and forwarding the data (per word ECC allows a word to be read, checked and forwarded immediately).

5. Provide stronger correction. ECC per word has lower probability for unrecoverable/undetectable error when multi-bit errors occur and can correct errors distributed in different words. This is one of the reasons for employing column interleaving in SRAMs.

REA behaves exactly the same way on writes and reads as any baseline ECC scheme. REA is distinct in how it decodes a syndrome. Specifically, when a word is read and the syndrome indicates an error, REA proceeds to produce the syndrome of the other correlated locations and then the decoder processes all the syndromes together to decide the nature of the error.

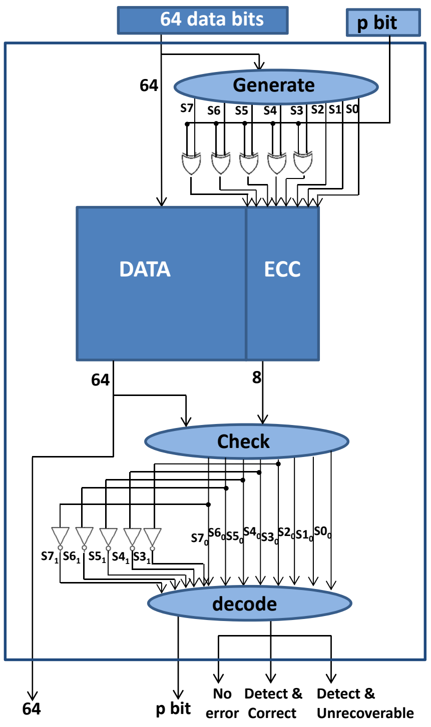


Figure 8: Overhead for Implicit-Storing 1-bit

Fig. 9 illustrates the generic concept of REA and how it operates upon detecting a fault. It reads multiple correlated locations and produces their syndromes. The syndrome decoder uses the multiple syndromes to decide how to react. Hence, REA can be classified as a n-way modular redundant scheme. When data is read, if there are errors, up to n-1 more syndrome generations are performed by reading the n-1 correlated locations. We discuss the performance implications of accessing correlated words in the memory hierarchy in Section 4.3

REA does not require writing the blocks in their entirety. Individual words may be updated separately as long as the attribute used for updating them is the same with their other correlated words. Also, when a block is filled the attributes of the correlated words need to be the same. These requirements ensure that correlated locations without faults have the same attributes.

Next we discuss how to combine REA with ECT and IS to recover some of the code-strength reduction they suffer from encoding additional info in their codewords.

4.1 Explicit and Redundant Encoding of Attributes (EREA)

EREA is a combination of ECT and REA that aims to recoup some of the code-strength loss of ECT. EREA behaves exactly the same way on writes and reads as a baseline ECT scheme (see Section 2.4). EREA is different from ECT in how it reacts to error detection.

We present the EREA operation with the help of an example case that considers correlation across two words and one attribute bit. The EREA behavior with more correlated words and attribute errors has also been derived but not shown due to space limitations. In particular, Fig. 10 presents the EREA reaction to different number of errors when an attribute is redundantly encoded and stored in two

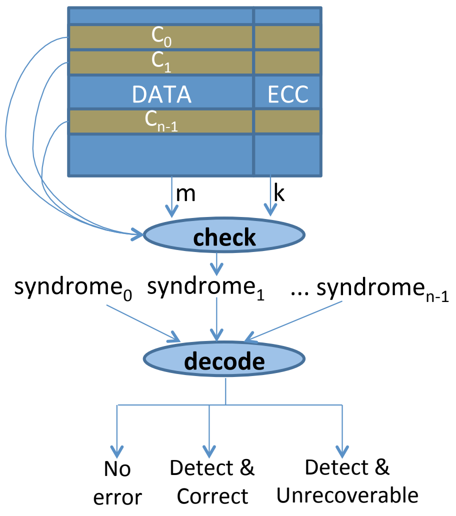


Figure 9: Concept of REA Producing Syndromes of Correlated Words

correlated locations C0 and C1. Fig. 10(a) shows the behavior when data errors occur only in location C0 whereas Fig. 10(b) presents the behavior with data errors in both locations. Redundant-encoding ensures that when an attribute error occurs it will appear in both correlated words. Assuming we read from either of these locations and an error is detected, then the two syndromes of C0 and C1 will be produced and they will be decoded as shown in Fig. 10. We note that some symmetric combinations of events are missing, such as data errors only in C1 instead of C0 or the 0-1/0[2] event, because the behavior for them can be mapped to other cases already present in Fig. 10.

First some remarks about the labels used in the figure. When syndromes with odd number of 1s are produced they can correspond to a data, attribute, or non-legal bit position (see Section 2) and, therefore, we can differentiate between the three cases as Odd Data, Odd Attribute and Odd3 respectively. For example, in Fig. 10(a) when we have 2 data errors only in C0 and 1 attribute error, both correlated words will produce a syndrome with odd number of 1s. However, the C0 will be different from C1 because it is the result of three faults whereas C1 is due to a single fault (for more why this happens check [12]). This and other similar distinctions are important to be able to uniquely identify the different cases. The reaction to an attribute error is shown as XYZ because it varies depending on what type of attributes are encoded. For example, if the attribute is a security ID, then the action XYZ will most likely be a reaction to a security violation, e.g. reset the system. In Fig. 10 we assume that attribute errors have higher priority and that is why when both types of faults are detected in a column the action XYZ is used.

It is noteworthy that EREA is able to always decode correctly the case with two data errors in one correlated word (Fig. 10(a)) something that the baseline ECT scheme could not provide always (Section 2.4). This shows that for this error scenario EREA is able to recover the full code strength lost due to ECT. Furthermore, for some of the 2-1/1 and 2-2/1 cases, where ECT performs miscorrection, EREA reacts correctly by detecting an attribute error. However, both EREA and ECT are pessimistic for all cases with 2-1/0 and 2-2/0 errors by detecting an attribute error whereas in reality there is an unrecoverable data error. Finally, both schemes perform a miscorrection in some cases with 2-2/1 errors. The cases where EREA does not improve over ECT correspond to the shaded cells in the Global Action row of Fig. 10(b).

The above confirms that EREA can improve the fault-coverage of ECT. However, the exact amount of improvement depends on the probability of multiple-bit upsets [8, 26] in correlated words. With current DDRx DRAM interfaces the probability of a fault resulting in a correlated error is significant [26]. The analysis based on multiple bit upsets distribution is important, but beyond the scope of this work because it is dependent on many challenging to model parameters. For instance, the parameters for SRAM caches include: the cause of the multiple bit upsets, such as energetic particles or voltage of operation [8], technology parameters, such as feature size [8] and well orientation [29], and array implementation details, such as degree of interleaving [13].

4.2 Implicit and Redundant Encoding of Attributes (IREA)

This section describes how to combine IS and REA to provide Implicit and Redundant Encoding of Attributes (IREA). The goal of this scheme is to provide the benefits of IS, increasing the logical cache capacity without increasing the physical size, but with a stronger code.

IREA operates identically on writes and reads as IS (Section 3). The key difference, similar to the ECT+REA combination, is that the decoding is not done per word but by using the syndromes of multiple correlated words. Recall that correlation exists when the two or more words in a block share the same attribute and, therefore, the same attribute value is implicitly-stored in all the correlated words. This redundancy is leveraged by the decoder of the IREA to minimize the code strength reduction that IS experiences in some cases with two data errors.

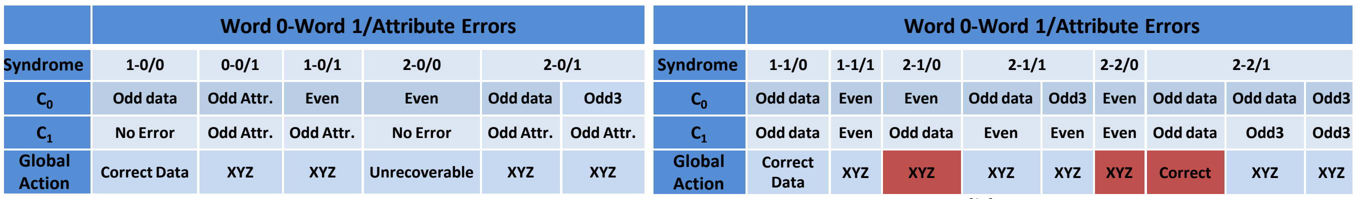
Fig. 11 presents the IREA decoder behavior with different number of faults in two correlated locations when a single bit attribute with value X is redundantly encoded in two correlated locations C0 and C1. The inputs of the decoder are the two syndromes from each location assuming the implicit value is X and X'. The figure shows what value is inferred for the implicit bit - X, X', or unknown (?)- and what action is taken for each of the words: No error (N), Correct (C), and Unrecoverable (U).

The table clearly shows that IREA is able to infer almost always the correct value for the attribute and also detect correctly what error is suffered by each word in the correlated pair. In the case of 1/2 and 2/1 errors the value of p cannot be inferred but the decoder recognizes correctly that one of the two words experiences an unrecoverable error. Depending on the use scenario, this information may be sufficient because an unrecoverable error will initiate some abort sequence.

It is important to note that in the case of two errors in one word (cases 2/0 or 0/2) the decoder recognizes that there is an unrecoverable error unlike in Fig. 7. This demonstrates that IREA recovers the strength code reduction of IS for that case.

The only problematic IREA case, for the errors combinations considered in Fig. 11, happens when two errors occur in each word. This corresponds to the last column in Fig. 11.

[2] x-y/z denotes x data errors in C0, y data errors in C1, and z attributes errors affecting both.



(a)

	Word 0-Word 1/Attribute Errors					
Syndrome	1-0/0	0-0/1	1-0/1	2-0/0	2-0/1	
C_0	Odd data	Odd Attr.	Even	Even	Odd data	Odd3
C_1	No Error	Odd Attr.	Odd Attr.	No Error	Odd Attr.	Odd Attr.
Global Action	Correct Data	XYZ	XYZ	Unrecoverable	XYZ	XYZ

(b)

	Word 0-Word 1/Attribute Errors								
Syndrome	1-1/0	1-1/1	2-1/0	2-1/1		2-2/0	2-2/1		
C_0	Odd data	Even	Even	Odd data	Odd3	Even	Odd data	Odd data	Odd3
C_1	Odd data	Even	Odd data	Even	Even	Even	Odd data	Odd3	Odd3
Global Action	Correct Data	XYZ	XYZ	XYZ	XYZ	XYZ	Correct	XYZ	XYZ

Figure 10: (a) EREA behavior with errors in one correlated word (C0) (b) EREA behavior with errors in both correlated words

The wrong value of p is misinterpreted and both words are miscorrected.

Quantifying the exact strength-code improvement of IREA vs IS is quite challenging and dependent on many parameters, as argued in Section 4.1. We, nonetheless, quantify analytically the probabilities for spatial and temporal multi-bit error to illustrate how REA can help improve the code strength when considering random independent single-bit transient faults. It is stressed that the expected improvement from the use of REA depends on the probability distribution of multi-bit upsets in correlated-words which are not examined in the following analysis.

The following analysis measures how likely it is for two bit flips to occur in a word as compared to two flips in each word of a correlated pair. First we quantify analytically these two probabilities for spatial multi-bit error assuming a cache with n 64 bit words that is protected with a 73-65 SEC-DED code with 1 implicit bit.

For IS the probability for the miscorrection in a cache is given by:

$$P_{IS} = 1 - (1 - p_w)^n \tag{1}$$

where p_w is the probability of a word to experience a miscorrection from a two bit error:

$$p_w = \binom{72}{2} pfail^2 (1 - pfail)^{72-2} W_3 \tag{2}$$

where *pfail* is the probability for a single bit flip, and W_3 is the probability when two bits are flipped and the attribute value is wrong to have miscorrection. We obtained W_3 to be 0.2827 when using a Hsiao 72-64 SEC-DED code extended with an unused weight-5 column that minimizes W_3 to represent the implicit bit.

For IREA the probability for a miscorrection in a cache is given by:

$$P_{IREA} = (1 - (1 - {p_w}^2)^{n/2}) \tag{3}$$

The number of words is divided by two because IREA treats words in pairs and the probability for two words to experience miscorrection is the product of each to experience a miscorrection.

We also present the temporal analysis using PARMA [28] that estimates the probability of IS and IREA to experience a miscorrection. The probability for IS miscorrection in a cache is given by:

$$P_{IS_t} = AVF * p_{wt} * n \tag{4}$$

where AVF is the probability that a fault will affect the result of the program. We assume an AVF of 90%. Also p_{wt} is the probability for a word to experience a miscorrection in t vulnerability cycles, where t is the average number between two consecutive accesses to a word. The p_{wt} can be obtained from:

$$p_{wt} = \binom{72}{2} p_t^2 (1 - p_t)^{72-2} W_3 \tag{5}$$

where p_t is the probability of at least one bit flip to happen in t vulnerability cycles:

$$p_t = (1 - (1 - pfail)^t) \tag{6}$$

For IREA the probability for temporal miscorrection in a cache is given by:

$$P_{IREA_t} = AVF * p_{wt}^2 * \frac{n}{2} \tag{7}$$

The number of words is divided by two for the same reason it is done for the spatial analysis.

Fig. 12 shows the ratio of spatial (P_{IS}/P_{IREA}) and temporal (P_{IS_t}/P_{IREA_t}) as a function of pfail for different cache sizes. The results demonstrate that IREA can minimize the IS strength reduction by many orders of magnitude underlying the potential of the REA approach to mitigate code-strength reduction.

4.3 Performance Overheads of EREA and IREA

The redundant-encoding employed by EREA and IREA requires in some cases to access multiple correlated words. In this Section we discuss the implications of accessing correlated words on performance of L1 data caches, non-L1 caches and main memory, assuming correlation between pairs of words. The discussion considers the behavior with no-errors and correctable errors since for the other remaining error types the recovery methods are more geared for functional correctness than performance (e.g. restart after detecting an unrecoverable error).

The operation and performance of a L1 data cache that uses EREA remains as usual until an error is detected in a word. In such case, to resolve the type of error, EREA requires another access to read the correlated word to determine depending on the two syndromes the type of error and the action to take. The latency from the initial detection to the eventual resolution can be a handful of cycles in modern processors but if correctable errors occur rarely this performance overhead can be ignored. In the case where correctable errors occur frequently, to avoid any performance degradation, one may allow the speculative forwarding of the corrected value, using the one codeword, while waiting for the eventual resolution. In the rare case where the final outcome indicates an uncorrectable or attribute error EREA can leverage existing pipeline flushing mechanisms in the microarchitecture to prevent this data from polluting the architectural state.

The use of IREA for a L1 data cache seems inappropriate since it may require an access to the correlated word on

		Data Errors Word 0/Word1															
Syndrome		0/0	0/1	0/2		1/0	1/1	1/2		2/0		2/1		2/2			
C_0	**X**	No Error	No Error	No Error	No Error	Odd	Odd	Odd	Odd	Even	Even	Even	Even	Even	Even	Even	Even
	X'	Odd	Odd	Odd	Odd	Even	Even	Even	Even	Odd3	Odd	Odd3	Odd	Odd3	Odd3	Odd	Odd
C_1	**X**	No Error	Odd	Even	Even	No Error	Odd	Even	Even	No Error	No Error	Odd	Odd	Even	Even	Even	Even
	X'	Odd	Even	Odd3	Odd	Odd	Even	Odd3	Odd	Odd	Odd	Even	Even	Odd3	Odd	Odd3	Odd
Infer p value		**X**	**X**	**X**	**X**	**X**	**X**	**X**	**?**	**X**	**X**	**X**	**?**	**X**	**X**	**X**	**X'**
Global action		**NN**	**NC**	**NU**	**NU**	**CN**	**CC**	**CU**	**U?**	**UN**	**UN**	**UC**	**U?**	**UU**	**UU**	**UU**	**CC**

Figure 11: Individual and Global Syndrome Decoding assuming 2-way IREA with 1 implicit bit

each write access to ensure the coherence of implicitly-stored information in correlated words. This implies doubling write accesses to the L1 data cache which can be detrimental to performance, energy and power.

For non-L1 caches and main-memory an access typically involves a whole block. In a typical processor with deep cache hierarchy the practice is to bring an entire block from lower cache (e.g. L2) to upper cache (e.g. L1) on a read operation or on a replacement to read the replaced block when dirty (for writeback caches). So in both cases the entire cache block is read. Therefore, both EREA and IREA need no extra access for correlated words since correlated pairs belong to the same block and when accessing one word its correlated word will also get accessed. Therefore, the only performance overhead can come from having a word to wait for its correlated word to be accessed.

When using EREA for non-L1 caches and main-memory there is no delay when no error is detected and if correctable errors are frequent the latency overhead can be mitigated by forwarding speculatively values. Note that some controllers allow forwarding values before getting checked for integrity and are capable of forwarding subsequently the correct value or take an exception in the case of uncorrectable error. EREA can leverage such mechanisms to mitigate such performance overheads.

For IREA there is also no latency overhead in the case of no-error for non-L1 caches and main-memory. As shown in Fig. 11, for all cases that a word with no data error is accessed one of its two syndromes, X or X', indicates no error and the correct value of the implicit bit can be inferred unambiguously. However, when both syndromes of a word detect some error, the error can not be resolved unambiguously until the correlated word is accessed. For example, when the two syndromes of word, X and X', are Odd and Even then the correlated word is needed to determine what action to take. For instance, if the correlated word syndromes, X and X', indicate No Error and Odd then the error can be corrected and the implicit value can be inferred. However, if the syndromes are Even and Odd, the implicit value cannot be inferred and there is an unrecoverable error.

For caches that access correlated words in parallel IREA may not suffer any performance overhead for correctable errors. However, main memory and some caches have narrow transfer interfaces that limit access to a word at a time and, therefore, expose the delay between accessing the words in a correlated pair. If correctable errors are rare such performance overhead is negligible, but if they are frequent then this IREA overhead may degrade the performance.

Another possible overhead of redundant-encoding is the extra state needed when correlated words are not accessed in parallel. Specifically, the first word accessed in pair as well as its syndrome need to be stored until its correlated word is accessed and decoded at which time the two syndromes can be used to determine what action to perform.

5. APPLICATION AREAS

Herein, we describe some of the potential application areas of the proposed schemes.

5.1 ECT and EREA Uses

ECT and EREA can be used to enhance the protection of caches and memory with location specific invariant information that is available every time a location is accessed. Their main benefit comes from eliminating the overheads from storing the attribute information in the array. Examples of such invariant information are address bits [4] and security ID. In the case of the address, a hash or a subset of the address bits of a block can be used as attribute information. This information is encoded on a write and checked at read time to determine whether both the data and address are correct. In the case of error detection, ECT and EREA can localize where the error occurred: data, peripheral logic (address decoders, muxes etc) or both. Depending on the type of error a different recovery action can be initiated.

ECT and EREA can also be used to encode a security ID. For example, a location may be marked by a security ID and only be accessible when the provided security ID does not cause an error. When an error is detected due to a wrong security ID it can lead to a freeze or a shutdown. Since ECT and EREA do not strictly check whether the ID is identical but rather that its encoding is correct, such an approach may be better suited for low cost platforms that aim to provide inexpensive lightweight security.

When ECT and EREA are used in arrays where a read accesses an entire block and the attribute is multi-bit and common across all words in a block, the attributes can be encoded as follows. The attribute can be split and each part encoded in different words in the block. This means the check of the entire attribute is accomplished with many checks. Splitting the attribute may be preferable over hashing the entire attribute into few bits and encoding it in all words, because it can reduce the probability of undetected errors due to the aliasing caused by hashing.

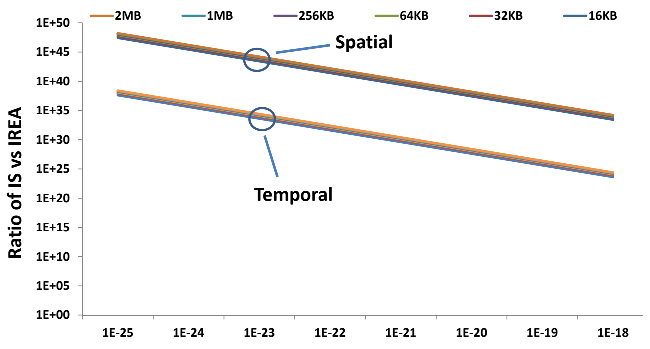


Figure 12: Ratio of Spatial and Temporal Probability for Miscorrection of IS vs IREA assuming 2-way Redundant Encoding of an implicitly stored bit

5.2 IS and IREA Uses

IS and IREA can be thought as inexpensive methods to tag information to words in a cache and main memory. Consequently a variety of previously proposed techniques that require some extra bit or bits per word or block can potentially benefit from the proposed approach. All the uses, therefore, that we describe next are feasible by using extra bits in the cache but this would entail larger area, more energy and possibly longer latency.

One possible use of IS and IREA is to track the dirty status in write-back caches at a finer granularity than an entire block. For example, words in a block can implicitly encode whether they are dirty. This information can be propagated through the cache hierarchy and can offer fault-tolerance, bandwidth, energy and performance benefits [30].

Without IS and IREA when a block is read from the L2 and it is both dirty and includes an unrecoverable fault, it may cause a halt, crash or revert to a checkpoint (if one exists). However, if the block is clean we can get its copy from a lower-level in the hierarchy. Tracking dirty status at finer granularity helps increase the potential to recover from otherwise unrecoverable faults. The finer tracking of dirty blocks also means less bandwidth, energy and potentially better performance since when we evict dirty blocks only the dirty subblocks need to be updated. For this application of IS and IREA, it may be advantageous to track the dirty bits explicitly in the L1 cache and have them implicitly in the rest of the hierarchy. This may be desirable to avoid performance degradation due to the need to update the dirty bit of all correlated words in L1 when one of the word changes. This is not an issue for writeback caches below L1 where accesses are typically performed at block granularity.

Another use of IS and IREA is for taint analysis [27]. In taint analysis it is desirable to track the flow of information through memory but this requires extra bits at a fine granularity in the cache hierarchy. IS and IREA can be used to implicitly tag words in a block without the storage overhead.

One other application of IS and IREA is to facilitate the lazy resolution of unrecoverable errors [31] using poison bits that are propagated through the memory hierarchy. This is used to track the influence of an error and only cause an exception when it will otherwise result in a user visible failure.

Finally, implicit-storing can be used to implement cost-effectively tagged main memory to improve the performance of graph-oriented applications [17].

5.3 Selective Use and Architectural Support

We anticipate that ECT, IS, EREA and IREA will be built in cache and memory controllers of future processors and architectural support will be provided to determine whether they are enabled and in what configuration. The selective use allows to assign to the same ECC hardware different capabilities that programmers and system can exploit in different situations. The configuration can be selected at boot time or during operation. The second is more flexible but may be a bit intrusive since it requires re-encoding the array contents with new (or no) attributes dynamically.

Let's illustrate the selective use with a hypothetical example. A system with EREA can disable REA if the aim is to maximize protection of data. But, if the goal is to protect against both data and attribute errors then EREA can be enabled. In another situation EREA can be disabled during normal mode but enabled when the hardware starts experiencing faults. By getting EREA enabled it facilitates the debugging process and helps better diagnose the problem source.

Additionally, a processor with IREA capability may be used more efficiently if a new data type is supported that permits tagging [10, 17] and operations on tags. IREA, therefore, can be used both for error protection and to improve analysis and performance.

In general, programmers and system should be informed about the implications on the fault-coverage of different configurations (e.g. how code strength is influenced as a function of attribute bits used and configuration).

6. AREA, DELAY, ENERGY AND SCALABILITY ANALYSIS

This Section quantifies the area, delay, energy and scalability trade-offs of the proposed schemes for caches and main memory.

We have measured the cache data array access delay, area and energy numbers for a cache implemented in a 32nm Low Power(LP) process using Artisan Memory Compilers [1]. Artisan Memory Compilers, which are offered by ARM, provide various silicon-proven SRAM, Register File and ROM memory compilers for CPUs and SoCs ranging from performance critical to cost sensitive and low power applications. We have also written the ECC logic design in Verilog, synthesized and implemented in 32nm LP libraries by using the *Synopsys IC* compiler implementation flow. The propagation delay, dynamic and static power consumption of the post-layout ECC logic circuit design are measured using the *Synopsys PrimeTime* suite.

Our analysis shows that 99.1% of the energy is due to reading the block from the data array and 0.9% due to the ECC logic for the data array of a 2MB, 16-way, 64B per block serially-accessed L2 cache using ECC with (72-64) Hsiao code. For every extra bit that is added per 64-bit word, the area and energy increase roughly by 1.25% per bit whereas delay roughly increases by 0.2% per bit. The overhead of single bit ECT, IS, EREA or EREA on ECC logic energy is insignificant. We have synthesized various ECC decoder designs under the same timing constraint and we are able to meet the delay constraint at the expense of minimal overall cache area and energy increase. Note that the entire ECC logic contribution to the overall cache energy is less than <1% and an increase in the ECC decoder energy has minimal impact on the overall energy. For this work we, therefore, assume that the extra decoding logic needed can be added without affecting the critical path.

Considering the large real estate caches occupy in modern processors area savings of the order of few percent, such as those offered by IREA, translate to more chips per wafer, better yield due to smaller area and more profit. This is in addition to any other benefits obtained using the proposed schemes.

The cache area savings is a function of how many attribute bits are encoded in a word, how many words store this information redundantly, and the size of each codeword. For example, if a single bit is redundantly encoded in two 72-bit codewords the area savings are in the order of 0.7%. The area savings appear minor but they come with minimal cost, a handful of gates. Note that the relative area and energy savings depend on code strength and codeword length. For example for 39-32 SEC-DED the savings are expected to be close to 2.5% per redundant implicit bit.

The maximum cache energy and area savings of EREA and IREA is around 3.75% when 3 bits are implicitly/explicitly-stored for the 2MB configuration. With 3 bits there is no compromise in fault-tolerance when faults are limited in one of the correlated words. Going beyond 3 bits compromises the SEC-DED capability of the code.

The savings of the proposed schemes are more pronounced and critical when applied to main memory. The main memory is built using commodity cards and devices with fixed width and compliant to standards. This leaves little flexibility to introduce an extra memory device to store attributes and to modify DRAM protocols. For example, to store from one up to the width of a memory device attributes requires adding an extra device. For a memory using 72-64 SEC-DED this will translate to 11.1% (5.5%) more overhead for x8(x4) devices, a custom DIMM card with 10(19) devices and modifying the DRAM communication protocol. All this overhead and complexity are eliminated with the use of the proposed approaches.

Regarding scalability, we have mentioned in Section 4.2 that the logical number of IREA checkers grows exponentially with the number of erased bits: 2 checkers for 1 bit, 4 for 2 etc. However, for the SEC-DED code used in the paper the logic required to generate all syndromes is 8 inverters. The syndrome is produced as in the baseline and to produce the 8 syndromes, for the 8 possible values of p in the case of 3 implicit-bits, we select each syndrome bit in its true or complement form depending on how the values of the implicit-bits affect the parity of each syndrome-bit.

7. RELATED WORK

Most modern processors have some form of protection for values in architectural arrays such as cache tag and data RAM arrays. What is less known, is that processors both in high availability systems but also embedded processors protect arrays against address errors [21, 33, 2]. This can be useful for catching errors that result in accessing correct data in the wrong block. These previous works do not describe exactly the method used to accomplish this but we anticipate that some variation of ECT is employed.

Gumpertz [10] describes how an ECT approach can be used to detect address errors that occur when reading from incorrect memory locations. As far as we know, Gumpertz's work is the first to explain why attributes that remain invariant between a write and read from a location need not be stored together with the data and can be used to augment protection by encoding them in the ECC code. However, in that work double bit data errors and single bit attribute errors result in identical type of syndromes (with even number of 1s in the case of Hsiao code) and the combination of a single bit data and a single bit attribute error can result in miss-correction. Therefore, the solution proposed in [10], that extends Hsiao code with ECT capabilities, is not Hsiao compliant and is not leveraging the extra space of the shortened code to detect errors.

It is of course possible to overcome some of the limitations of [10], in the case of a Hsiao code, by assigning each bit attribute an unused odd column (see Section 2). However, still the code will not be able to differentiate, in the case of double errors, whether the errors are only in data or they involve attribute bits.

Abella et al. [4] proposed to hardwire in each wordline its ID. Whenever, accessing a wordline the ID is read and compared with the index that is used to access the array to detect faults in the array peripheral circuits that lead to accessing an incorrect wordline (e.g. address decoders). This approach is able to detect address errors and differentiate between data and attribute errors but requires array modifications to hardwire the wordline ID.

Meixner et al. [23] proposed to detect data and address errors by storing the xor of the address and data of a location after the parity of the data is computed. When a value is read out it is xored with the address that is used to access the location and then its parity is computed. When there is an error in the data or address it can be detected. This scheme, however, is only limited to detection and is unable to differentiate between data and attribute errors.

Our work is distinct from the ECT scheme in that we redundantly encode the attribute in multiple correlated locations to facilitate distinction between faults in attributes and data, and therefore, avoid miss-corrections or undetected errors.

Erasure coding is widely used in different domains as a way to mitigate faults that may occur in the field and are readily identifiable [24]. Implicit-storing, as proposed in this work, is distinct in that it erases an attribute intentionally, without an error, aiming to save space.

The concept of using spatial redundancy for reliability is well known and widely used [18]. REA can be thought as a n-way modular redundant scheme that leverages the combined coding capacity offered by two or more codewords to increase the strength of a code.

8. CONCLUSIONS

This paper introduces two coding methods useful for caches and main memory protected with shortened ECC codes: implicit-storing and redundant-encoding-of-attributes. The first is useful to increase the logical capacity of a cache without increasing its physical size. The second is beneficial for recovering some of the strength of a code that is reduced due to the encoding of additional attributes in the code. Redundant-encoding augments the protection capabilities of an array by redundantly encoding the same attributes in multiple word locations. The proposed method, in general, does not incur storage cost but requires a slightly more expensive error decoding procedure sometimes involving multiple syndromes. Redundant-encoding is considered in combination with error-code-tagging and implicit-storing. Discussion of several uses underlines the potential benefits of the proposed approaches.

9. ACKNOWLEDGMENTS

The research leading to this paper is partially supported by the "EuroCloud, Project No 247779" and "Harpa, Project No 612069" of the European Commission 7th RTD Framework Program - Information and Communication Technologies: Computing Systems, and by the University of Cyprus. Yiannakis Sazeides completed part of this work during a HiPEAC (FP7 Network of Excellence) sponsored mini-sabba tical at ARM. Jaume Abella has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557. Also we like to thank the anonymous reviewers for their constructive critique and feedback that helped improve the paper quality.

10. REFERENCES

[1] Artisan Memory Compilers. www.arm.com/products/physical-ip/embedded-memory-ip,2013.

[2] Cortex-A9 technical reference manual. infocenter.arm.com, 2010.

[3] Cortex-r4 and cortex-r4f technical reference manual. infocenter.arm.com, 2010.

[4] J. Abella, P. Chaparro, X. Vera, J. Carretero, and A. González. On-line failure detection and confinement in caches. In *IOLTS*, pages 3–9, 2008.

[5] AMD Corporation. *BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors*, 2009. Order Number> 32559 Rev. 3.16 Nov. 2009.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *DAC '03*, pages 338–342, 2003.

[7] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *DAC46*, pages 4–7, New York, NY, USA, 2009. ACM.

[8] A. Dixit and A. Wood. The impact of new technology on soft error rates. In *SELSE11*, Mar. 2011.

[9] P. Elias. Coding for two noisy channels. In *The 3rd London Symposium, Information Theory*, pages 61–76, 1955.

[10] R. H. Gumpertz. Combining tags with error codes. In *ISCA*, 1983.

[11] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 26(2):147–160, 1950.

[12] M. Y. Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM Journal of Research and Development*, 14(4):395 –401, july 1970.

[13] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba. Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule. *IEEE Transactions, Electron Devices on*, 57(7):1527–1538, 2010.

[14] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*, 2013. Order Number> 253669-046US March 13.

[15] C. Keltcher, K. McGrath, A. Ahmed, and P. Conway. The amd opteron processor for multiprocessor servers. *IEEE, Micro*, 23(2):66 – 76, march-april 2003.

[16] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th International Symposium on Microarchitecture*, pages 197–209, Dec. 2007.

[17] S. Li, K. Chen, M. yu Hsieh, N. Muralimanohar, C. D. Kersey, J. B. Brockman, A. F. Rodrigues, and N. P. Jouppi. System implications of memory reliability in exascale computing. In *SC*, 2011.

[18] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200 –209, april 1962.

[19] M. J. Mack, W. M. Sauer, S. B. Swaney, and B. G. Mealey. Ibm power6 reliability. *IBM Journal of Research and Development*, 51(6):763 –774, nov. 2007.

[20] M. Manoochehri, M. Annavaram, and M. Dubois. Cppc: correctable parity protected cache. In *ISCA 38*, pages 223–234, 2011.

[21] C. McNairy and R. Bhatia. Montecito: a dual-core, dual-thread itanium processor. *IEEE, Micro*, 25(2):10 – 20, 2005.

[22] C. McNairy and D. Soltis. Itanium 2 processor microarchitecture. *IEEE, Micro*, 23(2):44 – 55, march-april 2003.

[23] A. Meixner, M. E. Bauer, and D. J. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. *IEEE Micro*, 28(1):52–59, 2008.

[24] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD Conference*, pages 109–116, 1988.

[25] W. Peterson and E. Weldon. *Error Correcting Codes.* MIT Press, 1972.

[26] V. Sridharan and D. Liberty. A study of dram failures in the field. In *SC*, 2012.

[27] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. *SIGARCH Comput. Archit. News*, 32(5):85–96, Oct. 2004.

[28] J. Suh, M. Manoochehri, M. Annavaram, and M. Dubois. Soft error benchmarking of l2 caches with parma. In *SIGMETRICS*, pages 85–96, 2011.

[29] A. Tipton, J. Pellish, J. Hutson, R. Baumann, X. Deng, A. Marshall, M. Xapsos, H. Kim, M. Friendlich, M. Campola, C. Seidleck, K. LaBel, M. Mendenhall, R. Reed, R. Schrimpf, R. Weller, and J. Black. Device-orientation effects on multiple-bit upset in 65 nm srams. *IEEE Transactions, Nuclear Science on*, 55(6):2880–2885, 2008.

[30] S. Wang, J. Hu, and S. Ziavras. On the characterization and optimization of on-chip cache reliability against soft errors. *IEEE Transactions, Computers on*, 58(9):1171 –1184, sept. 2009.

[31] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA 31, 2004.

[32] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA35*, pages 203–214, June 2008.

[33] A. Wood, R. Jardine, and W. Bartlett. Data integrity in HP nonstop servers. In *SELSE*, Apr. 2006.

[34] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J. L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T. J. O'Gorman, B. Messina, T. D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. R. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus. Ibm experiments in soft fails in computer electronics (1978-1994). *IBM Journal of Research and Development*, 40(1):3–18, 1996.