COMET: On-die and In-controller Collaborative Memory ECC Technique for Safer and Stronger Correction of DRAM Errors

Irina Alam and Puneet Gupta
Electrical and Computer Engineering
University of California, Los Angeles
USA
irina1@ucla.edu

***Abstract*—DRAM manufacturers have started adopting on-die error correcting coding (ECC) to deal with increasing error rates. The typical single error correcting (SEC) ECC on the memory die is coupled with a single-error correcting, double-error detecting (SECDED) ECC in the memory controller. Unfortunately, the on-die SEC can miscorrect double-bit errors (which would have been safely detected but uncorrected errors in conventional in-controller SECDED) resulting in triple bit errors more than 45% of the time. These are then miscorrected in the memory controller $>$55% of the time resulting in silent data corruption. We introduce COllaborative Memory ECC Technique (COMET), a novel method to efficiently design either the on-die or the in-controller ECC code, that, for the first time, will eliminate silent data corruption when a double-bit error happens within the DRAM. Further, we propose a collaboration mechanism between the on-die and in-controller ECC decoders that corrects most of the double-bit errors without adding any additional redundancy bits to either of the two codes. Overall, COMET can eliminate all double-bit error induced silent data corruptions and correct almost all (99.9997%) double-bit errors with negligible area, power, and performance impact.**

***Index Terms*—DRAM, Error Correction Codes, ECC, On-die ECC, Reliability, Bit-steering**

I. Introduction

With increasing rate of scaling induced errors in DRAM [4], [14], [22], [23], [26], [28], [34], [35], the traditional method of row/column sparing used by DRAM vendors to tolerate manufacturing faults [17] has started to incur large overheads. To improve yields and provide protection against single-bit failures in the DRAM array at advanced technology nodes, memory manufacturers have started incorporating on-die error correction coding (on-die ECC) [4], [31], [34]. The ECC encoding/decoding happens within the DRAM chip. The parity bits are stored in redundant storage on-chip and are not sent out of the chip; only the actual data, post correction, is sent out of the DRAM, making on-die ECC transparent to the outside world. Though DRAM manufacturers do not usually reveal their on-die ECC design and implementation, prior works [28], [31]–[34] and industry whitepapers [4] indicate the most commonly used scheme is (136,128) Single Error Correcting (SEC) Hamming code [16]. This code corrects any single-bit error that occurs in 128 bits of data with the help of 8 bits of additional parity. On-die ECC is typically paired with rank-level single error correction, double error detection (SECDED) code in the memory controller. Several server and desktop class processors implement in-controller SECDED ECC [2], [3], [12]. The main focus of in-controller ECC is to correct errors that are visible outside the memory chip, mostly due to failures in pins, sockets, buses, etc.

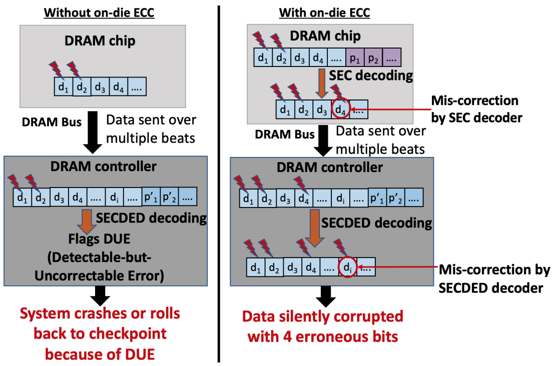

Without on-die ECC
DRAM chip
d_1 d_2 d_3 d_4 ....
DRAM Bus
Data sent over multiple beats
DRAM controller
d_1 d_2 d_3 d_4 .... d_i .... p'_1 p'_2 ....
SECDED decoding
Flags DUE (Detectable-but-Uncorrectable Error)
System crashes or rolls back to checkpoint because of DUE
With on-die ECC
DRAM chip
d_1 d_2 d_3 d_4 .... p_1 p_2 ....
SEC decoding
d_1 d_2 d_3 d_4 ....
Mis-correction by SEC decoder
DRAM Bus
Data sent over multiple beats
DRAM controller
d_1 d_2 d_3 d_4 .... d_i .... p'_1 p'_2 ....
SECDED decoding
d_1 d_2 d_3 d_4 .... d_i ....
Mis-correction by SECDED decoder
Data silently corrupted with 4 erroneous bits

Figure 1: Example showing the difference when a DBE occurs in DRAMs with and without on-die SEC. Both systems have in-controller SECDED. Assumption: data and parity bits that get decoded in the controller in one cycle are sent from the same DRAM chip across multiple beats.

The on-die SEC code covers for all the single-bit errors (SBE) [4], [8] within the DRAM chip. However, with incessant memory density scaling, the rate of double-bit errors (DBE) within the memory array is growing [23], [28], [34]; so DBEs are no longer a rarity. However, a double error correcting (DEC) code would incur large overheads and negate some of the density scaling benefits. As a result, it is not practical for DRAM manufacturers to have on-die DEC mechanism. In today's high reliability systems, the rank-level in-controller ECC is expected to detect DBEs and the system then restarts or rolls back to a checkpoint [11]. With an on-die SEC engine alongside in-controller SECDED engine, the data pipeline now includes two error correcting engines. This unfortunately increases the chances of silent data corruption (SDC) when a DBE occurs in a memory array. Let us explain this using an example.

As shown in Figure 1, without on-die SEC, the data goes through a single round of decoding inside the memory controller where the SECDED decoder flags the DBE. Now, with on-die SEC, the data first goes through the SEC decoder that only ensures protection against SBEs. For DBEs, this SEC decoder has a $>45\%$ (on average based on 10 random SEC constructions) chance of miscorrection resulting in a triple-bit error. The resulting data with the triple-bit error, when sent through the in-controller SECDED decoder, has a $\sim 55\%$ (on average, based on 10 random SECDED constructions) chance of being falsely considered as a SBE. The SECDED decoder would then miscorrect and silently corrupt the data. For a raw bit error rate of 10^{-4} that is often seen in recent works and industrial studies [8], [23], [28], [29], [34], we can expect SDC once every $\sim$ 300,000 SECDED decoding cycles in a system with a single DRAM chip that has on-die (136,128) SEC and in-controller (72, 64) SECDED. Thus, on-die ECC actually worsens memory reliability when DBEs occur in the memory array. So, we first ask the question: *Can we eliminate DBE-induced SDCs by careful construction of either of the two codes?* **In this work, we, for the first time, provide a framework to efficiently design either the on-die SEC code or the in-controller SECDED code in order to eliminate silent data corruption with no additional parity bits.**

Our next observation is, for every 128-bits of data, with the two ECC schemes combined, we now have 8 more bits of parity (from on-die ECC) as compared to having only in-controller ECC. When do these extra 8-bits really help? They help only in the rare case when a single-bit fault outside the memory array (e.g. link/pin failure) coincides with an SBE in the chip. In this case, the in-controller ECC sees only the single-bit flip introduced by the external fault and is able to correct it. Other than that, the on-die SEC is not improving protection on top of what the in-controller code was already doing. So, we also asked the question: *With extra 8-bits of redundancy, can we enhance memory reliability?* **We further enhance the on-die SEC construction and propose a collaborative mechanism to now correct majority of the double-bit errors that occur inside the memory array.**

This work, Collaborative Memory ECC Technique (COMET), makes the following key contributions:

- We provide a detailed on-die SEC code construction technique (SEC-COMET) that completely eliminates DBE-induced SDCs at no additional parity storage overhead. The design technique exploits the overall memory system architecture and steers the miscorrected bit when a DBE occurs in such a way that the in-controller SECDED, irrespective of its implementation, never encounters all three bits of errors in the same decoding cycle, thereby guaranteeing no SDC. For SEC-COMET to be effective against SDCs, no special in-controller SECDED construction is required. The two codes (on-die and in-controller) can be constructed independently.
- If the DRAM manufacturer does not guarantee on-die SEC-COMET, we show how the in-controller SECDED can then be specially designed to take care of SDCs. We provide a detailed construction of the SECDED code for a given on-die SEC implementation and memory system architecture.
- We develop a collaborative DBE correction technique. The SEC code needs to be designed with an additional constraint and the memory controller needs to send a special command with additional information once a detectable-but-uncorrectable error (DUE) is flagged. This collaborative technique can correct almost all (99.9997%) DBEs while ensuring no miscorrections.
- SEC-COMET implementations require no additional parity bits, have less than 5% decoder area and latency overheads and less than 10% decoder power overhead as compared to the most efficient SEC construction.

II. Background

This section provides an overview of DRAM operation, coding theory background and comparison of ECC codes seen in today's DRAM-based memory subsystems.

A. DRAM Operation

Dynamic Random Access Memory (DRAM) chip cell stores a single bit of data in a capacitor [9], [10]. These cells are organized in two dimensional arrays called banks. A read/write command accesses a small subset of columns in a row and includes multiple steps. First the entire row is read into a row buffer using the ACTIVATE command. Then a READ/WRITE command is sent with the column address to initiate the data transfer. An x*N* DRAM chip uses N data pins (DQs) in parallel during data transfer [25], [39]. Typically, more than one DRAM chip is accessed in parallel to improve bandwidth and they together form a rank. A single memory access takes multiple cycles – during each cycle a *beat* of data (*N* bits from every chip in a rank) is transferred. The number of beats transferred in each access constitutes the memory burst length. The number of cycles per access and the width of a data beat accessed in each cycle depends on the memory device and the data access protocol. If a rank consists of 8 x8 DRAMs and the burst length is 8 beats, it translates to 64-bits of data transfer per beat and a total of 64B transfer per READ/WRITE command.

B. Linear Hamming Error Correcting Codes

Error correcting code (ECC) detects and/or corrects by adding redundant parity bits to the original data. A (n,k) Hamming code protects a k-bit dataword (original data) by encoding the data through linear transformation to form a n-bit codeword. The number of parity bits is equal to $n-k$. Increasing the number of parity bits increases the minimum Hamming distance between two legal n-bit codewords. A code of minimum distance d_{min} is guaranteed to correct $t=\lfloor\frac{1}{2}(d_{min}-1)\rfloor$ erroneous symbols. The encoding is done by multiplying the dataword ($\vec{m}$) with the generator matrix $\mathbf{G}$: $\vec{m}\mathbf{G}=\vec{c}$ and the resulting codeword $\vec{c}$ is written to memory. When the system reads the memory address of interest, the ECC decoder hardware obtains the *received codeword* $\vec{x}=\vec{c}+\vec{e}$. Here, $\vec{e}$ is an error-vector of length n that represents where memory faults, if any, have resulted in changed bits/symbols in the codeword. The decoder multiplies the received codeword $\vec{x}$ with parity check matrix $\mathbf{H}$ to calculate the *error syndrome*: $\vec{s}=\mathbf{H}\vec{x}^{\mathrm{T}}$. The following conclusions can be drawn from the syndrome:

- s = 0: No error.
- s $\neq$ 0: Error detected; syndrome is matched with columns of the parity check matrix **H** to determine the exact bit-location of the error. If the syndrome matching is unsuccessful, the decoder declares it as a DUE.

The syndrome is generated without any knowledge about the exact number of errors in the received codeword. If the number of errors exceeds the correction capability of the code and the syndrome matching is successful it would mean one of the following scenarios have occurred:

- s = 0: The decoder declares the codeword error-free and all bits of errors go undetected.
- s $\neq$ 0 and points to a bit: This bit can be one of the erroneous bits or a non-erroneous bit. In either case, the decoder will flag a CE and miscorrect that bit.

This leads to SDC where the decoder wrongly declares data with errors as correct. In this work, we attempt to reduce such SDC events when double-bit errors occur.

C. SEC vs. SECDED

Single-Error Correcting (SEC) codes ($d_{min} = 3$) correct all possible SBEs. The columns in the parity-check matrix **H** of a linear SEC code are distinct and the minimum number of columns to form a linearly dependent set is 3. This ensures that every legal codeword is at-least 3 bit flips away from each other. Single-Error Correcting, Double-Error Detecting (SECDED) codes ($d_{min} = 4$) [18] can correct all SBEs and detect all possible DBEs. The minimum number of columns to form a linearly dependent set in the parity-check matrix **H** of a linear SECDED code is 4. Every legal codeword is at least four bit flips away from each other. Both these codes can correct SBEs. In case of DBEs, SEC code either declares a DUE or miscorrects by flipping a third bit. SECDED, on the other hand, always declares a DUE when a DBE occurs.

III. Motivation

Single bit errors are still the majority of the failures in today's DRAMs. Hence, DRAM manufacturers have started adopting on-die ECC for better reliability. Based on our system level reliability analysis (details provided in Section VI-B) we see that on-die SEC ECC helps to reduce system failures by more than 35%. However, it is ineffective for multi-bit errors and instead introduces unexpected miscorrection.

A. Miscorrections by On-Die ECC

Let us consider an example of a DRAM device with the most commonly used (136,128) SEC Hamming code. This SEC code can correct any SBE. However, in case of a multi-bit error, there are two possible outcomes: (1) The errors go undetected and is equivalent to not having an on-die ECC mechanism. (2) The multi-bit error aliases to a single-bit error. This happens when the sum of the columns in the H-matrix of the decoder corresponding to the error positions is equal to another column in the matrix.

The second case is the problematic one. In order to better understand this case, consider the following example SEC $\mathbf{H}_{\text{example}}$ parity-check matrix with 128 message bits and $r = 8$ parity bits:

$$\mathbf{H}_{\text{example}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{array} \begin{array}{c} \begin{array}{cccccc|cccccccc} d_1 & d_2 & d_3 & d_4 & d_5 ... d_{127} & d_{128} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array} \\ \left[\begin{array}{cccccc|cccccccc} 1 & 0 & 0 & 1 & .... & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & .... & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & .... & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .... & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .... & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & .... & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & .... & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & .... & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array},$$

where d_i represents the ith data bit, p_j is the jth redundant parity bit and c_k is the kth parity-check equation. Now, if a double-bit error occurs in bits 1 and 2, the resulting codeword c' is equivalent to adding error patterns e_1 and e_2 to the original codeword c. e_i is a 136-bit vector with bit i = 1 and all other bits = 0. By the definition of a linear block code, $H.c = 0$ for all legal codewords c. Therefore, error patterns e_1 and e_2 isolate columns 1 and 2 of the SEC H matrix (i.e., $\mathbf{H}_{example_{*,1}}$ and $\mathbf{H}_{example_{*,2}}$) and as shown in Equation 1, the resulting syndrome is the sum of the two columns.

$$\begin{aligned} &\mathbf{s} = \mathbf{H}_{\text{example}} . \mathbf{c}' = \mathbf{H}_{\text{example}} . (\mathbf{c} + \mathbf{e_1} + \mathbf{e_2}) \\ &= \mathbf{H}_{\text{example}} . \left(\mathbf{c} \quad + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) = 0 + \mathbf{H}_{example_{*,1}} + \mathbf{H}_{example_{*,2}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \mathbf{H}_{example_{*,4}} \end{aligned} \tag{1}$$

Now, the sum of columns 1 and 2 of the $\mathbf{H}_{\text{example}}$ matrix is equal to column 4. Therefore, the generated syndrome **s** matches column 4. As a result, the decoder would consider it as a single bit error in bit position 4 and flip it as part of its correction mechanism. Thus, an originally double-bit error has now become a triple-bit error. On an average (across 10 random SEC Hamming code constructions), the chances of a DBE miscorrecting to a triple bit error is >45%. With increasing DRAM error rates, recent studies [8], [23], [29], [34] have shown that the probability of a DBE occurring within the 128-bit dataword can be as high as $\sim 8 \times 10^{-5}$, which translates to a DBE every 12500 SEC decoding cycles. Thus, the chances of a double-bit error converting to a triple-bit error are also high and will only increase in the future.

B. SDC post in-controller SECDED decoding

Now let us look at the problems that arise because of this miscorrection. SECDED code inside the memory controller is not designed to detect more than double-bit errors. As a result, when the (136,128) SEC on-die ECC miscorrects and converts a DBE to a triple-bit error, there is a high probability (greater than 50% on an average over multiple SECDED codes) for the SECDED decoder to consider it as an SBE and further miscorrect. This will happen when the generated syndrome or the sum of three columns in the SECDED parity check matrix corresponding to the erroneous bits is equal to a fourth column. The probability of SDC depends on the exact SECDED code and the memory data transfer protocol. A widely used on-die ECC is (136, 128) SEC [4], [31] and in-controller ECC is (72, 64) SECDED [24]. For the rest of the paper, we will use these two codes for explaining our proposed code construction mechanisms and DBE correction technique. However, our proposed constraints can be easily

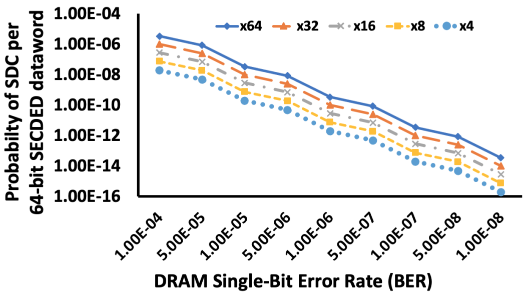


Figure 2: Probability of SDC every 64-bits of SCEDED dataword read from memory when a double-bit error occurs in a system with (136,128) on-die SEC and (72,64) in-controller SECDED coding schemes for different bit error rates and data access protocols.

extended to other SEC and SECDED code constructions with different dataword and codeword lengths.

DBEs are becoming more probable with increasing bit error rate in the recent DRAM generations. Multiple recent experimental/industrial studies [8], [23], [29], [34] have considered DRAM raw bit error rate (BER) as high as 10^{-4}. For different memory system architectures and data access protocols, we evaluate the probability of SDC when a DBE occurs for BERs ranging from 10^{-4} to 10^{-8}. The result is shown in Figure 2. For this evaluation we consider the average miscorrection rate across ten different (136,128) on-die SEC and (72,64) in-controller SECDED implementations. We evaluate for different access protocols; x64 means all 64-bits of SECDED dataword come from the same DRAM chip while x4 means there are 16 DRAM chips and each DRAM chips sends 4-bits per beat of memory transaction. For a BER of 10^{-4}, the probability of silent data corruption in the case of x16 data access protocol is non-negligible and can happen once every 3 million 64-bit accesses. As the data width per chip reduces, the SDC probability decreases. This is because the probability of a DBE, along with the miscorrected bit, aligning perfectly within the same beat boundary reduces with decrease in beat width. Without on-die SEC, the SDC probability, however, is 0 since all double-bit errors within the DRAM array, irrespective of location, would not get miscorrected and would be flagged as DUE by the in-controller SECDED decoder. Thus, while the SEC code does not help with detecting or correcting the double-bit errors in any scenario, it causes miscorrection and turns upto 25% of these DBE events into SDC.

IV. Designing ECC to Eliminate SDC

In today's DDR or LPDDR based systems, during every read operation, the data that is read into the memory controller is typically striped across multiple DRAM dies. Each xN DRAM die sends N-bits data in parallel during each *beat* of memory transfer to construct the 72-bit controller codeword. But inside each DRAM chip, the SEC decoding is performed on a much larger 128-bit dataword. Only a part of this 128-bit data is accessed by the memory controller per read (see Figure 3). This mean that the dataword of on-die ECC gets striped across multiple in-controller SECDED codewords. This has significant implications on SDC probability. The DBE probability in a 128-bit word and the SEC-induced miscorrection rate remain constant across the DRAM dies having the exact same SEC implementation. However, the probability of the double-bit error and the miscorrected bit coinciding within the same in-controller 64-bit dataword decreases with the decrease in the amount of data from each on-die dataword that constitutes the in-controller codeword (as shown in Figure 2). If all 64-bits come from the same DRAM chip and, therefore, from the same 128-bit SEC dataword, the SDC probability is $> 150x$ higher than the case where 16 x4 DRAM chips send 4-bits each in parallel.

In this work, we provide two possible solutions that exploit this data access pattern to completely avoid SDCs when DBE occurs. (1) An on-die SEC construction technique which ensures that the miscorrected bit is steered to a different beat. It does *not* require knowledge of the exact in-controller code and is compatible with any SECDED implementation in the controller. (2) Our alternate solution outlines an in-controller SECDED construction technique that ensures that none of the on-die aliasing triplets result in SDC. However, unlike the first technique, this in-controller SECDED construction needs to know the on-die SEC code.

A. On-die SEC-COMET ECC

In this work we exploit the data transfer protocol in DRAMs to take care of SDCs. As shown in Figure 3, if all the three erroneous bits in the 136-bit codeword do not get transferred and decoded in the memory controller in the same beat, the SECDED decoder will not encounter a triple-bit error and SDC can be avoided. Thus, the on-die SEC has to be constructed such that the miscorrection from any DBE within a single beat gets steered to a bit position that belongs to a different transfer beat. This will ensure that the three erroneous bits never coincide in the same 72-bit SECDED codeword.

In order to achieve this property in a (136, 128) SEC code, within every beat transfer boundary, the sum of any two columns in the parity check **H** *matrix should not be equal to a third column in the same set.*

Step-by-step code construction and mathematical guarantee: With 8-bits of parity per 128-bits of dataword, the COMET-SEC additional constraint can be satisfied when designing the SEC code for any data transfer protocol as long as the beat transfer boundary (N) consists of 64-bits (64 columns) or less. When constructing the 8×136 parity check matrix H, we can choose the 136 8-bit columns from 128 odd-weight and 127 even-weight non-zero options. The DBE-induced miscorrection happens when the sum of two columns is equal to a third column in the H matrix. Either all these three columns would have even weights or two of them would have odd weights and the third would have even weight. Thus, the two aliasing sets possible are: (1) (odd, odd, even), (2) (even, even, even). If we could construct the H matrix with all odd weight columns then no pair of columns would sum up to a third column and there would be no DBE-induced miscorrection. However, we do not have enough odd-weight columns for the entire matrix. Hence, when constructing the H matrix for our proposed SEC-COMET code for an xN DRAM architecture, we use a two-step approach.

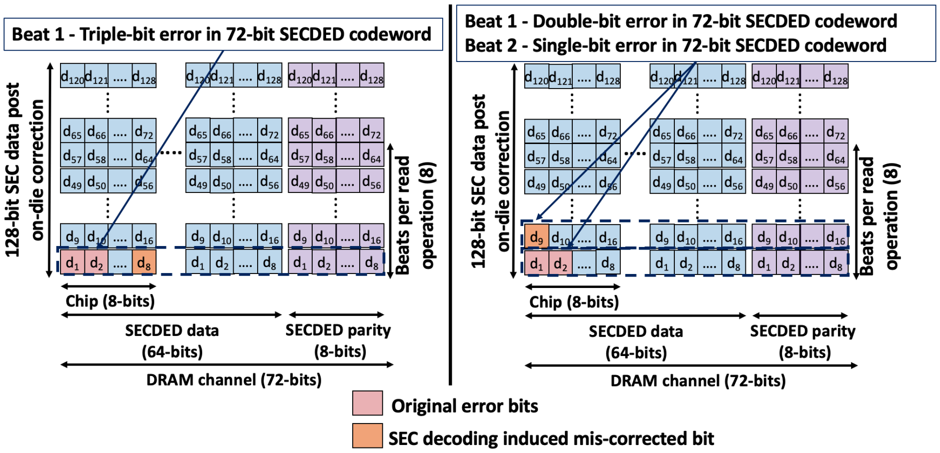
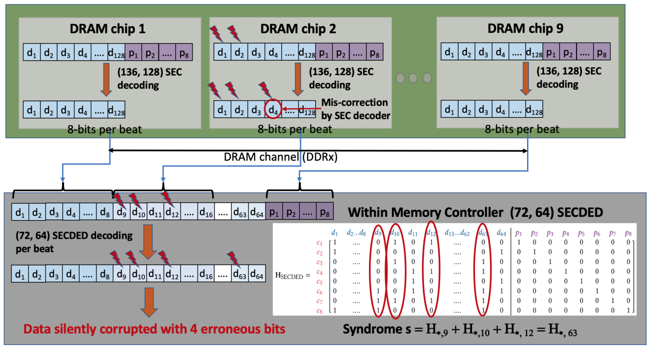

Beat 1 - Triple-bit error in 72-bit SECDED codeword
128-bit SEC data post on-die correction
Beats per read operation (8)
Chip (8-bits)
SECDED data (64-bits)
SECDED parity (8-bits)
DRAM channel (72-bits)
Beat 1 - Double-bit error in 72-bit SECDED codeword
Beat 2 - Single-bit error in 72-bit SECDED codeword
128-bit SEC data post on-die correction
Beats per read operation (8)
Chip (8-bits)
SECDED data (64-bits)
SECDED parity (8-bits)
DRAM channel (72-bits)
Original error bits
SEC decoding induced mis-corrected bit

Figure 3: Example showing how steering the miscorrected bit to a different beat transfer boundary during SEC decoding prevents the SECDED decoder from encountering the problematic triple-bit error within the same 72-bit codeword.



$$H_{SECDED} = \begin{array}{c|cccccccccc|cccccccc} & d_1 & d_2...d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13}...d_{62} & d_{63} & d_{64} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \\ \hline c_1 & 1 & .... & 0 & 1 & 0 & 1 & .... & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_2 & 1 & .... & 0 & 0 & 0 & 0 & .... & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_3 & 0 & .... & 0 & 1 & 0 & 0 & .... & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ c_4 & 0 & .... & 0 & 0 & 1 & 1 & .... & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ c_5 & 0 & .... & 0 & 0 & 1 & 0 & .... & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ c_6 & 0 & .... & 1 & 0 & 1 & 0 & .... & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ c_7 & 0 & .... & 1 & 1 & 0 & 1 & .... & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ c_8 & 1 & .... & 1 & 0 & 0 & 0 & .... & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Syndrome $s = H_{*,9} + H_{*,10} + H_{*,12} = H_{*,63}$

Figure 4: Example showing SDC occurring due to miscorrection introduced by on-die ECC. We have considered the SEC construction provided in Section III-A where the sum of columns 1 and 2 in the $H_{example}$ matrix is equal to column 4.

- Out of the 128 odd weight columns, we use the single weighted columns for the last 8 identity sub-matrix columns. We use the remaining 120 odd-weight options for the first $min(128-N,120)$ columns. None of these columns would have the problem of aliasing since they are all odd.
- For the remaining $X = max(N,8)$ locations, we use only even weight columns. We randomly choose a bit position (say bit 0) and set it to '1' for all X columns. If bit 0 for all X columns is '1', then the sum of any pair of columns cannot equal a third column in this set as bit 0 of the sum would always be '0'.

The total number of such even weight columns possible $= \binom{7}{1} + \binom{7}{3} + \binom{7}{5} + \binom{7}{7} = 64$. Therefore, N can be as wide as 64 (largest possible factor of 128). Thus, on-die SEC-COMET code can be constructed for x4 to x64 DRAMs that can *guarantee* no silent data corruption. Note that this SEC-COMET construction requires no knowledge of the in-controller SECDED code.

B. In-controller SECDED-COMET ECC

Now we ask the question: *In case the DRAM device does not implement our proposed on-die SEC-COMET ECC provided in Section IV-A, can we still tackle the problem of SDCs from the memory controller side?* In this section we provide a technique to redesign the in-controller SECDED code, albeit with the knowledge of the SEC code used in the memory device. A recent work [34] has proposed an efficient way of reverse engineering the exact on-die SEC implementation.

Once we know the SEC code, we will know all the bit positions pairs (H column pairs) that lead to a miscorrection (sum of columns equal to third column) within the same beat transfer boundary. These are the triplets that eventually can lead to SDC. We first list all such bit positions triplets. In $\mathbf{H_{example}}$ provided in Section III-A, one such bit position triplet is: positions 1, 2 and 4. This is because sum of columns 1 and 2 equal to column 4 and all three bit positions lie within the same beat transfer boundary. For every triplet, we then calculate all possible corresponding bit positions in the SECDED dataword. For example, in an x8 DRAM architecture, bits 1, 2, and 4 in the SEC dataword can correspond to any of the following bit positions (in their respective order) in the SECDED dataword (spanning 8 DRAM chips):

- Bits 1, 2 and 4 - Chip 1
- Bits 9, 10 and 12 - Chip 2
- ....
- Bits 57, 58 and 60 - Chip 8

This is because the 64-bit SECDED dataword that is decoded in the memory controller in each beat spans across 8 DRAM chips. Hence, bit 1 of the SEC dataword from chip 1 would be bit 1 of the SECDED dataword, but bit 1 of the SEC dataword from chip 2 would be bit 9 of the SECDED dataword. The same is true for the rest of the DRAM chips. We need to consider positions corresponding to all DRAM chips since they all would use the same on-die SEC code.

Now let us consider the example shown in Figure 4. A DBE affects bits 1, 2 in chip 2 and bit 4 gets miscorrected by the SEC decoder. Post data transfer, this translates to triple-bit error in bit positions 9, 10 and 12 in the SECDED codeword. This becomes an SDC since the sum of these columns in the SECDED H matrix is equal to another column (column 63 in the example). The decoder flips bit 63, declares the error correction as a success, and sends the corrupted data over to the processor. In order to prevent this SDC from happening in a system with this particular on-die SEC code, the SECDED parity check matrix has to be designed such that the sum of all the sets of columns corresponding to the bit positions listed above do not match with any of the columns in the rest of the H matrix. To construct such an H matrix, we first assign any three 8-bit non-zero values to columns 1, 2 and 4. We then store the sum of these three columns in a don't-use list. For the next set of three columns (9, 10 and 12), we again choose three random non-zero values that do not equal the value stored in the don't-use list. We then sum these three columns and append the sum to the don't-use list. We continue the process for the rest of the columns.

The process has to be repeated for all bit triplets in the SEC dataword that lead to three-bit errors in the final SECDED codeword. Once all bit triplets are covered, the remaining SECDED H-matrix columns are randomly assigned 8-bit values that do not match anything stored in the don't-use list. *For a given SEC code and system memory architecture, for every bit/column triplet in the SECDED* **H** *matrix that can cause SDC, the sum of the columns has to be such that it equals no other column in the* **H** *matrix.*

Using this technique, given the exact SEC implementation and the system architecture, it is possible to construct the SECDED code that would guarantee prevention of SDCs when double-bit errors happen.

V. COMET Double-bit Error Correction

As mentioned previously, on-die ECC adds 6.25% parity storage overhead without improving error correction capability. Previous studies have shown that there is almost no difference in reliability between DIMMs with 8 chips that have only on-die ECC and DIMMs with 9 chips that support both on-die ECC and rank-level in-controller SECDED ECC [28]. Thus, the two disjoint ECC schemes together do not reduce the overall system failure probability. Instead we have shown that, if one of them is not carefully designed, it causes additional SDCs. So we next try to answer the question: *Can we achieve better reliability from the two codes while keeping the two code types the same and their constructions independent?*

In this section, we show how DBE correction can be achieved with no extra parity overhead using the redundancy built within the two codes. We add one more constraint to the on-die SEC code construction and devise a controller-device collaborative correction mechanism to get nearly perfect double-bit error correction. *It is important to note that even though the collaborative technique requires controller-device communication using a special command, the two ECC codes can be designed completely independently and does not require any special in-controller SECDED construction.*

A. Constructing on-die SEC code to enable Double-bit Error Correction (SEC-COMET-DBC)

In order to enable detection and correction of DBEs using syndrome matching we need to ensure that the sum of any pair of columns in the parity check matrix **H** generates a unique syndrome. However, with just 8-bit redundancy for a 128-bit dataword, this can be achieved only for a small subset of columns. We add a constraint to SEC-COMET code construction from Section IV-A to construct the SEC-COMET-DBC code: for every set of x consecutive columns, the sum of every pair of columns within that set should be unique. For a (136, 128) SEC code, the maximum value of x (that is also a factor of 128) for which this can be possible is 16. I.e., a valid SEC-COMET-DBC code can be constructed for x4, x8, x16 DRAM chips but not for x32. This is because, for every pair of columns to generate a unique syndrome in a set of 32 columns, $\binom{32}{2} = 496$ unique syndromes are required. This is not possible with 8-bits.

For such a SEC code, when a double-bit error occurs in bit positions that belong to the same x-bit chunk, the generated syndrome and the chunk position can be used to figure out the exact DBE locations. The syndrome is generated by the SEC decoder, but for the correction mechanism to work, the errors also have to be localized to the exact x-bit chunk which the SEC decoder is unable to do. For this localization we will exploit the memory data access architecture and utilize information from the in-controller SECDED decoder. For example, in a standard x8 DDR based ECC DIMM, the beat transfer width per chip is 8 and therefore, we use $x = 8$ in the (136, 128) SEC-COMET-DBC code. Now when a DBE happens within the same 8-bit chunk in one of the DRAM chips, the beat in which the decoder flags a DUE will help to point to the 8-bit chunk position where the DBE has occurred. Next, we discuss how this information can be sent to the DRAM chips and the the DBE correction flow. For better understanding we explain the mechanism using a x8 DDR architecture.

B. Collaborative DBE Correction

1) Detecting the DBE beat

Let us look at all the possible ways a double-bit error can happen in a 136-bit codeword in a particular DRAM chip and the possible outcomes after the on-die and in-controller decoding.

- Case 1: The two error bit positions are in two different 8-bit chunks and the miscorrected bit (if any) belongs to a third chunk. As a result the erroneous bits get decoded in the memory controller in separate beats. In each of these beats, the SECDED decoder flags a CE and corrects the error. Eventually all the erroneous bits get corrected and no DUE gets flagged.
- Case 2: The two error bit positions are in two different 8-bit chunks and the miscorrected bit falls in the same chunk with one of the error bits. Now one 8-bit chunk that has two errors and one has single-bit error. The in-controller SECDED decoder will flag a CE when it decodes the chunk with SBE but will flag a DUE when the 8-bit chunk with two error bits is decoded.
- Case 3: The two error bit positions are in the same 8-bit chunk. The SEC-COMET constraint (provided in Section IV-A) will ensure that the miscorrected bit lands in a different 8-bit chunk. Thus, after SEC decoding the 128-bit dataword either has one 8-bit chunk with two errors (in the case of no miscorrection) or has an additional 8-bit chunk with a single-bit error. The SECDED decoder will flag a DUE when the 8-bit chunk with two error bits is decoded.

Let us consider the example shown in Figure 3 (Case 3). A DBE occurs in DRAM chip 1 in bits 1 and 2. Because of our improved SEC construction (shown on the right), it is ensured that the SEC decoder would steer the miscorrection to a different 8-bit chunk (in this example the miscorrected bit is 9). Therefore, during the first beat of memory transaction, the SECDED decoder flags a DUE, while in the second beat it flags a CE and corrects bit 9. The memory controller uses a special error correction command to send the original read command address and the beat number in which the DUE was flagged to the DRAMs. The SECDED decoder cannot localize the DBE to a particular chunk in the codeword. Therefore, the double-bit error could have occurred in any of the 9 DRAM chips. Every DRAM chip receives the information that there might be a DBE in the first 8-bits of its 128-bit SEC dataword.

2) *Correction within each DRAM chip*

Once the memory controller sends the special double-bit error correction command with the beat number, each DRAM chip checks the original SEC syndrome. We assume that the special DBE correction command immediately follows the original READ command. Therefore, the DRAM chips only need to store the last generated 8-bit syndrome and the 32-bit/64-bit data that was last read. Storing the original data has negligible overhead but prevents an extra ACTIVATE during correction and possible change in error signature in case of closed page policy. If the syndrome was zero, the DRAM knows that the DBE did not occur in its codeword. In our example (Figure 3), all DRAMs except chip 1 would have generated a zero syndrome. If the syndrome is non-zero, the correction mechanism within the chip tries to match the syndrome with one of the **H** matrix columns in the 8-column set that corresponds to the received beat number. In this case, DRAM chip 1 tries to match the syndrome against columns 1-8 (beat 1) in the H matrix. We know that the miscorrected bit position is 9. Therefore, the generated syndrome would match with column 9. Since, this column falls outside the target set, the matching is unsuccessful. The decoder moves on to the next step where it matches the generated syndrome with the sum of every pair of columns from the target set. Because of our improved SEC construction, every pair of columns should sum up to a unique value. The pair of columns whose sum equals the generated syndrome (in this example it will be columns 1 and 2) represent the erroneous bit positions. The decoder would flip those two bits and send the corrected data over the DRAM bus to the memory controller. The rest of the DRAM chips would not take any action since they had zero syndrome and send the original 8-bit data.

While the example depicts Case 3, let's look at what happens in Case 2. In this scenario, the original double-bit errors are in two separate beat transfer chunks. But the miscorrected bit lands in the same 8-bit chunk as one of the two errors. Let's say this is the second 8-bit chunk. Thus, the SECDED controller flags DUE in the second beat and sends this information to the DRAM chips. When the erroneous chip matches the generated syndrome against columns 9 to 16 in the **H** matrix, it sees that the syndrome matches with the column corresponding to the miscorrected bit position. In this case, the DRAM chip would only flip that particular bit and send over the data to the DRAM controller. It will not be able to localize and correct the second error position within that 8-bit chunk. Considering the rest of the DRAM chips had zero syndrome, they send their unmodified data over in the same beat. Since the erroneous chip could only correct one bit, the overall data still has one-bit of error that SECDED will be able to correct.

3) *Final Correction within the memory controller*

The final correction step in the DRAM controller involves multiple rounds of SECDED decoding of the corrected data. This is to provision for the rare cases where DBE in one chip coincides with SBEs in other chips within the same 8-bit chunk. In that case, multiple DRAM chips would see non-zero syndromes that match with one of the columns in the target set. All these DRAMs would send data uncorrected corrupted data that would need to be filtered out on the memory controller side.

Once the controller receives the new 72-bit codeword from the DRAMs, it compares it with the one it had received during the original read. In the ideal case where only a single DRAM chip has DBE and no other chip has made any corrections, the two codewords would differ by one/two bits within a particular 8-bit boundary corresponding to the erroneous chip. However, in the rare case where multiple DRAM chips send modified data, the controller, post comparison, would find bit flips in more than one 8-bit chunk. The four possible multiple-erroneous-chip scenarios are shown in Figure 5 where one chip has a DBE and the other chip has a SBE. To prevent miscorrection and silent data corruption, the controller accepts changes corresponding to each chip (i.e., each 8-bit chunk) one at a time and sends the new data through the decoder. Let's say chips 1 and 8 send new data. The controller will first accept the change from chip 1, keep the old data from chip 8 and send the entire 64-bit data through the decoder. Then it will revert chip 1's change, accept the new data from chip 8 and send this new 64-bit data through the decoder. If one of these two cases result in zero syndrome (scenarios b and d), the controller declares the corresponding data as correct and moves ahead. If both cases return a non-zero syndrome but in one case the decoder detects and corrects a SBE and in the other case the decoder detects a DUE (scenario c), the controller declares the data corresponding to the first case as correct and moved ahead. If both cases result in non-zero syndrome with the decoder detecting an SBE (scenario a), the controller panics and declares the error uncorrectable. This rare scenario arises when DBE Case 2 (explained in Section V-B1) occurs in one chip and an SBE in another chip.

The likelihood of this uncorrectable case is $\sim$1 in 300,000 DBEs. I.e., COMET achieves 99.9997% double-bit error correction. The step-by-step correction mechanism of DBEs by COMET is shown in Figure 6. A similar correction outcome is expected if there is link error instead of single-bit error in the data signals of the other chips. The probability of double-bit error striking two different DRAM chips within the same beat transfer boundary is less than 2×10^{-10} with BER of 10^{-4}. Therefore, we only consider upto single bit error in the other DRAM chips.

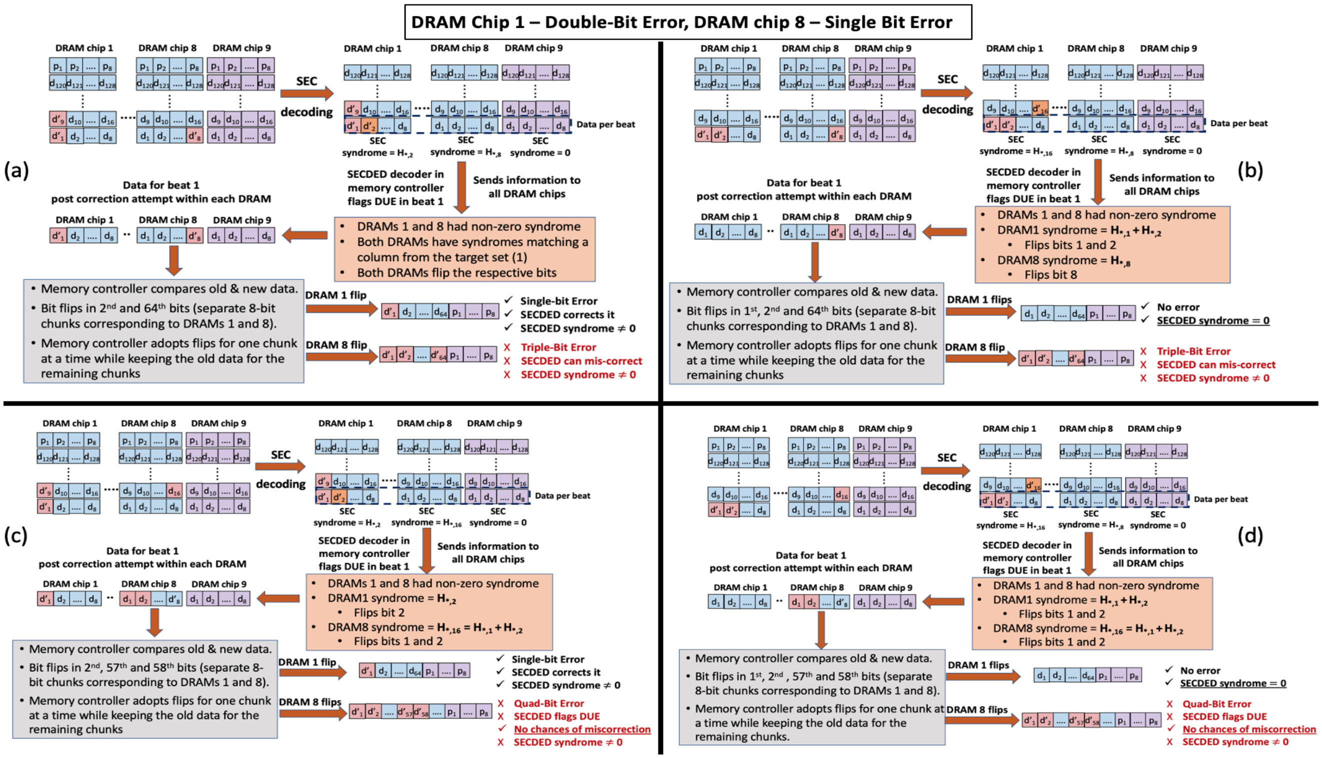

DRAM Chip 1 – Double-Bit Error, DRAM chip 8 – Single Bit Error
(a)
(b)
(c)
(d)
DRAM chip 1
DRAM chip 8
DRAM chip 9
SEC decoding
Data per beat
SEC syndrome = H*,2
SEC syndrome = H*,8
SEC syndrome = 0
SECDED decoder in memory controller flags DUE in beat 1
Sends information to all DRAM chips
DRAMs 1 and 8 had non-zero syndrome
Both DRAMs have syndromes matching a column from the target set (1)
Both DRAMs flip the respective bits
Data for beat 1 post correction attempt within each DRAM
Memory controller compares old & new data.
Bit flips in 2nd and 64th bits (separate 8-bit chunks corresponding to DRAMs 1 and 8).
Memory controller adopts flips for one chunk at a time while keeping the old data for the remaining chunks
DRAM 1 flip
DRAM 8 flip
✓ Single-bit Error
✓ SECDED corrects it
✓ SECDED syndrome ≠ 0
X Triple-Bit Error
X SECDED can mis-correct
X SECDED syndrome ≠ 0
SEC syndrome = H*,16
DRAM1 syndrome = H*,1 + H*,2
Flips bits 1 and 2
DRAM8 syndrome = H*,8
Flips bit 8
Bit flips in 1st, 2nd and 64th bits (separate 8-bit chunks corresponding to DRAMs 1 and 8).
✓ No error
✓ SECDED syndrome = 0
DRAM1 syndrome = H*,2
Flips bit 2
DRAM8 syndrome = H*,16 = H*,1 + H*,2
Flips bits 1 and 2
Bit flips in 2nd, 57th and 58th bits (separate 8-bit chunks corresponding to DRAMs 1 and 8).
DRAM 1 flips
DRAM 8 flips
X Quad-Bit Error
X SECDED flags DUE
✓ No chances of miscorrection
X SECDED syndrome ≠ 0
DRAM8 syndrome = H*,16 = H*,1 + H*,2
Bit flips in 1st, 2nd, 57th and 58th bits (separate 8-bit chunks corresponding to DRAMs 1 and 8).
Memory controller adopts flips for one chunk at a time while keeping the old data for the remaining chunks.

Figure 5: The different scenarios possible when one chip has double-bit error and another chip has single bit error that aligns in a way leading to multiple DRAM chips modifying data during DBE correction

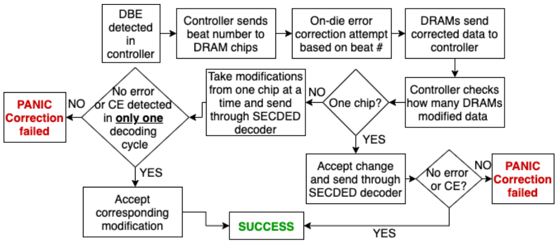

DBE detected in controller
Controller sends beat number to DRAM chips
On-die error correction attempt based on beat #
DRAMs send corrected data to controller
Controller checks how many DRAMs modified data
One chip?
NO
Take modifications from one chip at a time and send through SECDED decoder
No error or CE detected in only one decoding cycle
NO
PANIC Correction failed
YES
Accept corresponding modification
SUCCESS
YES
Accept change and send through SECDED decoder
No error or CE?
NO
PANIC Correction failed
YES

Figure 6: Step-by-step COMET double-bit error correction mechanism.

C. Implementation of COMET command

The DBE correction mechanism in COMET requires the controller to send a special correction command to the DRAMs to initiate the on-die correction. This command will need to send the exact beat number during which the DUE was flagged along with the rest of the column address. In DDR4/LPDDR4 standards, there are typically one or more spare command sequences that are reserved for future use (RFU). One such RFU command sequence can be used to support this special command.

In Table I we have listed a command sequence for DDR4 and LPDDR4 protocols that can be used for COMET DBE correction. In DDR4 it will be a single cycle single command sent on the rising edge of the clock while in LPDDR4 it will be a multi-cycle multi-command sent on successive rising clock edges like their standard read/write operations. In DDR4, address bits A[2:0] determine how the beats would be ordered when sending the data from a particular column address [19], [27] during a read operation. For example, A[2:0] = “010” would send beat number 2 first followed by beats 3, 0, 1, 6, 7, 4, 5, while A[2:0] = “101” would send beat 5 first followed by beats 6, 7, 4, 1, 2, 3, 0. The same address bits can be used in our special command to denote the target beat in which DUE had occurred and the DRAM device would correct and send data accordingly. Similarly, in LPDDR4 protocol [13], [20], C[4:0] of the 10-bit column address (C0 to C9) is used to determine the beat ordering during read operation and can be re-purposed in our special command to send the target beat number. Also, both protocols support burst chop, which allows the DRAM devices to send reduced number of beats during the memory transaction. Since we need only a single beat post correction from the DRAMs, the special command can enable burst chop. In DDR4, BC_n is set to LOW for a burst size of 4 beats instead of the standard 8 beats. In LPDDR4, the CA5 pin in the first cycle can be set to LOW for the shortest burst length. For DRAM devices that do not guarantee the COMET-SEC-DBC construction, the special command to correct double-bit errors can be turned off in the memory controller.

VI. Results

A. Reliability Evaluation

We evaluate the impact of double-bit errors and silent data corruption caused by these errors on system-level reliability through a comprehensive error injection study. While, in most cases, SDCs corrupt the final result or lead to unexpected crashes and hangs during the run of an application, some SDCs might get masked and would eventually have no impact on the final output. Since COMET

Table I: COMET DBE Correction Command Sequence in DDR4 and LPDDR4 protocols

DDR4								
Signals	**Clock Edge**	**Prev. CKE /Pres. CKE**	**CS_n**	**ACT_n**	**RAS_n/A16**	**CAS_n/A15**	**WE_n/A14**	**A[13, 11]**
COMET_special	R1	H	L	H	L	H	H	Valid Signal
Signals		**BG[1:0]**	**BA[1:0]**	**C[2:0]**	**A12/BC_n**	**A10/AP**	**A[2:0]**	**A[9:3]**
COMET_special		BG	BA	Valid Signal	L	L	Target Beat number	Column Address
LPDDR4								
Signals	**Clock Edge**	**CS**	**CA0**	**CA1**	**CA2**	**CA3**	**CA4**	**CA5**
COMET_special-1	R1	H	L	H	L	H	L	BL (L)
	R2	L	BA0	BA1	BA2	C0 (Target Beat)	C9	C1 (Target Beat)
COMET_special-2	R1	H	L	H	L	H	H	C8
	R2	L	C2 (Target Beat)	C3 (Target Beat)	C4 (Target Beat)	C5	C6	C7

ensures that none of the double-bit errors result in SDC, our objective is to understand the severity of on-die ECC induced SDCs in the event of a double-bit error without COMET in order to evaluate the usefulness of COMET.

We selected a random implementation of a (136, 128) SEC on-die code that obeys the basic constraints of a Hamming code and only ensures single-bit error correction. For the in-controller ECC, we selected a conventional (72, 64) Hsiao SECDED code [18] that is known to be widely used. Since approximation tolerant applications are expected to mask SDCs and be least impacted by them, we used benchmarks from the AxBench suite [44] for this study. Any standard approximation intolerant application is expected to strictly benefit more from COMET. We built AxBench against GNU/Linux for the open-source 64-bit RISC-V (RV64G) instruction set v2.0 [43] using the official tools [30]. Each benchmark is executed on top of the RISC-V proxy kernel [41] using the Spike simulator [42] that we modified to inject errors. We use our modified version of Spike to run each benchmark to completion 5000 times. During each run, a load operation is randomly chosen and a double-bit error is injected in a 128-bit word. The 128-bit SEC code decodes the erroneous codeword first, followed by the (72, 64) SECDED decoder. The chosen SEC and SECDED decoder combination has an overall 20.65% probability (average calculated across 100,000 random 136-bit codewords) of not flagging a DUE and resulting in a DBE-induced SDC because of miscorrections. We observe the effects on program behavior for the cases where DUE is not flagged and, therefore, corrupted data is sent over to the processor. The results are shown in Figure 7.

Overall, without COMET, on an average, ∼80% of the double-bit errors are flagged as DUE while less than 2% of the times the resulting SDC gets successfully masked by the application. ∼12%, on an average, result in erroneous output with a non-negligible impact on output quality and for the rest of the cases, the program either hangs or crashes. SEC-COMET or SECDED-COMET code constructions completely eliminate SDCs converting the unwanted output errors or crashes in the 18% of cases to more acceptable DUEs. SEC-COMET-DBC corrects nearly all of these double-bit errors, i.e., 98% point improvement in DBE reliability (no improvement in the 2% cases where the application masks the SDC caused by DBE).

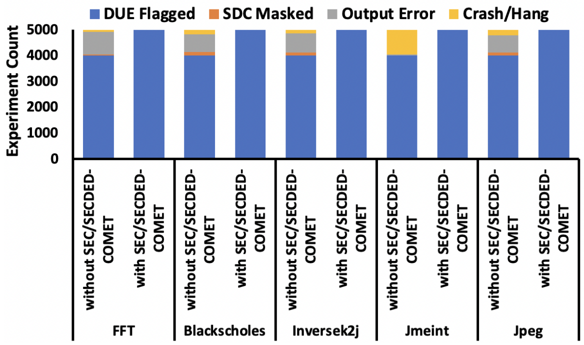


Figure 7: The impact of on-die ECC induced SDC (when running without SEC-COMET/SECDED-COMET) in the event of double-bit error on the program behavior when running applications from the AxBench suite.

B. *Effectiveness of COMET Double-bit Error Correction*

We evaluate the reliability of a system with 128GB DRAM with three different error correction schemes: no on-die ECC, standard SEC ECC and SEC-COMET-DBC scheme. We used fault simulator MEMRES [40] with real world field data from [37] and [40]. We took into account scaling induced bit error rate of 10^{-4} for this study. Our system has 2 channels, each containing dual ranked DIMM of 64GB capacity with 18 x8 DRAMs. In all three systems we have considered in-controller SECDED protection. We perform Monte Carlo simulations for a 5 year period and consider both undetected as well as detected-but-uncorrectable errors as system failures. For details on each failure mode, we refer the reader to [40]. Overall, we see that adding on-die SEC coding significantly helps in improving device failure by 35% over the system without any on-die coding. The main failure mode that on-die ECC takes care of is single bit permanent fault intersecting with a single-bit transient fault(SBT) in the array or the bus. The SBT in the array is taken care of by the occasional scrubbing that is enabled in the DRAMs. With scrubbing enabled, the DRAM dies, when idle, occasionally activate rows, check for errors in the row using the on-die SEC mechanism, correct (if possible) and write the data back. The intersection with bus faults is taken care by the on-die and in-controller ECCs. With COMET-SEC-DBC, we can achieve a 8.2% reduction in system faults over standard SEC, which translates to more than 150 lesser failures per year. This improvement in memory resiliency comes from double-bit correction which helps to reduce single-row failures and single-word failures.

Table II: Synthesis Results for Different x8 SEC Decoder Implementations in Commercial 28nm Library

	SEC-random	SEC-best case	SEC-COMET-DBC (x8)	SEC-COMET-DBC (x16)
Gate Count	168	165	170	170
Area (μm^2)	331.452	318.168	328.374	332.91
Latency (ps)	512	508	517	520
Power (W)	2.12E-05	1.93E-05	2.09E-05	2.12E-05

C. *Impact on Encoder/Decoder Area, Energy and Latency*

COMET code constructions do not require additional redundancy bits. But the encoder and decoder circuitry overheads varies based on the exact code implementation. The added constraints in COMET do not allow simplifications that could be done in other SEC codes to reduce encoder/decoder overheads. In order to evaluate our proposed SEC code overheads, we synthesized few different SEC implementations along with our construction using a commercial 28nm library. [1] We considered the SEC code with the minimum possible sum of the weight of the columns in the parity check matrix H as the most efficient implementation in terms of gate count. We also compared against a random SEC implementation which satisfies the basic Hamming code constraints required for single error correction. Based on the results in Table II, we see that the difference in area (<5%), latency (<2.5%) and power (<9.7%) among the different SEC decoders is minimal and negligible. Furthermore, on-die ECC consumes a very small fraction of the overall DRAM active power (∼5-7% [4]).

D. *Performance Impact of COMET schemes*

SEC-COMET/SECDED-COMET has no performance impact. SEC-COMET-DBC incurs additional latency only when a double-bit error occurs. In a system using x8 DDRx protocol based DRAMs with an elevated bit error rate of 10^{-4} and on-die (136, 128)SEC-COMET-DBC mechanism, a double-bit error in a 572-bit memory line that causes the (72,64)SECDED decoder to flag a DUE can happen once every ∼17,000 read operations. This is the probability of DBE occurring within a 136-bit SEC dataword where both error bits are either in the same 8-bit chunk belonging to the 64-bit half that is read from the chip or the mis-corrected bit coincides with one of the two erroneous bits. Only in this rare case, SEC-COMET-DBC uses additional cycles to correct the DBE. For all other read/write operations, SEC-COMET-DBC's encoding/decoding latency is comparable to any other on-die SEC code.

To evaluate SEC-COMET-DBC's DBE correction mechanism's impact on performance, we used cycle based simulation of 18 SPEC CPU 2017 benchmarks [1], 8 Parsec benchmarks [6] and 4 applications from the GAP suite [5] on the Gem5 simulator [7]. These are the applications that we could successfully compile and run using Gem5. We used a 2GHz single-core processor with a private 32KB I-cache, 64KB D-cache, shared 512KB L2 cache and shared 2MB L3 cache. For once every 17000 read operations, we doubled the read latency and added worst-case 9 memory cycle penalty for the DBE correction. We evaluated the DDR4-2400-x8 memory configuration with a 64b data channel for 2-billion instructions. The overall performance impact was less than 0.8% compared to an oracular case with no memory errors. This is because one additional memory read every 17k reads is still rare and has negligible impact on queuing delay and overall execution time. Of course, the impact on overall performance reduces with reduction in BER (<0.1% for BER of 10^{-8}). *Note that, in absence of SEC-COMET-DBC, these DBEs would require frequent checkpoint-recovery, the performance cost of which is extraordinarily high (30 minutes to restore a checkpoint [38]).*

VII. Discussion and Related Work

A. *Independent design of on-die and in-controller codes*

All three COMET schemes proposed allow within-DRAM SBE correction that is invisible to the rest of the system. Two of the schemes (SEC-COMET and SEC-COMET-DBC) allow independent code constructions by DRAM and CPU vendors. SEC-COMET and SEC-COMET-DBC require the DRAM vendors to add constraint(s) while constructing the on-die SEC. But the CPU vendors can design any SECDED code independently without requiring any knowledge of the on-die SEC implementation. We proposed in-controller SECDED-COMET for the case where SEC-COMET construction is not guaranteed by the DRAM vendor. SECDED-COMET guarantees protection from DBE-induced SDCs only for those DRAMs that have the on-die SEC implementation used for SECDED-COMET construction.

B. *Why Not Use Stronger On-die Codes?*

SECDED code [18] has the ability to detect double-bit errors, not correct them. Having on-die SECDED would prevent DBE-induced miscorrections. However, as DRAM vendors prefer check bits in multiples of 8 [32], the on-die ECC would be (72, 64)SECDED. This would double the parity storage overhead from 6.25% in (136, 128) SEC to 12.5%. Even after doubling the parity overhead, the code will only be capable of avoiding miscorrections due to DBE, it will not be able to correct the DBE. Our proposed COMET schemes have the same parity overhead (6.25%) as today's on-die SEC code while eliminating all DBE-induced SDCs and correcting almost all (99.9997%) DBEs. Thus, there is absolutely no advantage for using 2x higher parity overhead SECDED over COMET-SEC and hence, is not practical for DRAM manufacturers to prefer SECDED over SEC-COMET/SEC-COMET-DBC. On the other hand, double error correcting requires twice the number of parity bits per 128-bit of dataword while also significantly increasing the latency, area and power overhead of the encoder/decoder circuitry. As a result it would negate some of the density scaling benefits.

C. *Why Not Use Stronger In-controller ECC?*

Using a double-error correcting, triple error detecting (DECTED) scheme in the memory controller will require additional storage and data lines to transfer the extra parity bits. For every 64-bits of dataword, DECTED requires 7 extra parity bits as compared to SECDED. In some high

[1] Though DRAM technology is different compared to logic technology, the comparison between different implementations should still hold.

Table III: Comparison of different COMET schemes with past works

	On-die ECC check bits per 128b dataword	**In-controller ECC check bits per 64b dataword**	**Two codes designed independently**	**Memory protocol unchanged**	**Free of SDC caused by DBE**	**DBE correction per 64-bit of data**	**SBE correction in DRAM transparent to the system**
On-die SEC + in-controller SECDED	8	8	Yes	Yes	No	No	Yes
SEC-COMET	**8**	**8**	**Yes**	**Yes**	**Yes**	**No**	**Yes**
SECDED-COMET	**8**	**8**	**No**	**Yes**	**Yes**	**No**	**Yes**
SEC-COMET-DBC	**8**	**8**	**Yes**	**Special command**	**Yes**	**Yes (99.9997%)**	**Yes**
XED [28]	16	8	Yes	Yes	Yes	Yes	No
DUO VRT [15]	-	4	-	Extra bursts per access	Yes	One per 512b access	No
DUO SDDC [15]	-	12	-	Extra bursts per access	Yes	Yes	No
Minimal Aliasing SEC [32]	8	-	-	Yes	No (Reduced)	No	Yes

performance, high-reliability expensive systems today, single symbol correcting, double symbol detecting (SSCDSD, also known as Chipkill) coding is used to tolerate upto single chip failures. However, the standard 4-bit symbol Chipkill code used today can support only x4 DRAM chips [28]. In order to use x8 DRAM, one data access will have to be split into two, which will have a significant impact on performance. Entire chip failures are very rare and, therefore, Chipkill is considered an overkill in most systems today [28].

D. Comparison with Past Works

Several past works have proposed stronger memory reliability but most of them either do not improve on-die ECC or incur overheads and require changes to the standard protocol. Table III compares the COMET schemes with some of the related works. XED [28] proposes using error detection within each DRAM die and then exposing the detection result to the in-controller code for correction. But they assume that on-die codes implemented in today's DRAM have guaranteed double-error detection capability while in most known cases [4], the on-die code only guarantees single-error correction. Using the same code for multi-bit error detection will not be effective as the code would miscorrect. Besides, it does not support silent SBE correction within DRAMs which is desired by DRAM vendors. Similarly, DUO [15] also gets rid of on-die SBE correction and uses those additional bits for stronger in-controller protection. Thus, DRAM vendors cannot use DUO to improve yield. Besides, it requires non-negligible changes to the existing memory protocols. A recent work [32] highlights the aliasing problem in SEC codes and provides a construction technique that would result in minimal aliasing. However, their code would still result in SDCs (average $\sim$5 percentage points reduction in SDC probability when compared against 20 random SEC codes) when paired with in-controller SECDED unlike COMET that completely gets rid of SDCs by carefully steering the miscorrected bit. PAIR [21] uses on-die SECDED that requires N on-die ECC decoding cycles for xN DRAM. It ensures that each DQ bit comes from a separate codeword. This incurs a significant latency overhead and is not feasible for larger data width (x16/x32). Besides, it requires an additional signal to transfer the multi-bit error detection information. Other proposed reliability techniques like Bamboo-ECC [24] uses large ECC symbols and codewords to provide stronger protection while incurring performance overhead. ArchShield [29] provides protection against single-bit scaling induced errors but requires storing of fault maps within the DRAMs that would need to be updated in-field that requires running full array testing using a Built-In Self Test (BIST) engine. CiDRA [36] proposes using on-die ECC to provide protection against multi-bit failures. However, it requires large SRAM overheads that makes its usage prohibitive.

E. Accommodating Wider Data Widths

As mentioned previously in Sections IV-A and V-A, with 8-bits of parity for 128-bits of dataword, SEC-COMET (SEC-COMET-DBC) construction works upto per-chip beat width of 64 (16) bits. For wider interfaces, COMET cannot avoid SDCs or correct DBEs. To enable COMET, the 64-bit SECDED dataword has to be formed using multiple 128-bit SEC datawords. Therefore, within the DRAM chip, every 16-bits of the 64-bit data transferred needs to be a part of a different 128-bit SEC dataword. Thus, a single write or read command would require multiple rounds of on-die SEC encoding and decoding. Typically, during a read/write operation, an entire DRAM row gets activated into the row buffer. The size of a DRAM row is usually few kBs and therefore, contains multiple SEC datawords. Hence, to enable COMET for wider per chip beat widths, the multiple on-chip encoding and decoding can be done in parallel and would not require multiple activations of DRAM rows.

VIII. Conclusion

Aggressive technology scaling in modern DRAMs is leading to a rapid increase in single-cell DRAM error rates. As a result, DRAM manufacturers have started adopting on-die ECC mechanism in order to achieve reasonable yields. The commonly used on-die SEC ECC scheme interacts with in memory controller SECDED ECC, to unfortunately cause silent data corruption in >25% of double-bit-error cases. To prevent silent data corruption from happening, we introduce COllaborative Memory ECC Technique (COMET), a mechanism to efficiently design the on-die SEC ECC or the in-controller SECDED ECC that steers the miscorrection to guarantee that no silent data corruption happens when a DBE occurs inside the DRAM. Further, we develop the SEC-COMET-DBC on-die ECC code and a collaborative correction mechanism between the on-die and in-controller ECC decoders that allow us to correct the majority of the DBEs within the DRAM array without adding any additional redundancy bits to either of the two codes. Overall, COMET can eliminate all double-bit error induced SDCs and correct 99.9997% of all DBEs with negligible area, power and performance impact.

Acknowledgment

The authors thank the DSN'22 program committee and the anonymous reviewers for their detailed and constructive feedback. The authors thank Professor Onur Mutlu for being the shepherd of the paper and taking out time to review and provide valuable feedback.

References

[1] "SPEC releases major new CPU benchmark suite." [Online]. Available: https://www.spec.org/cpu2017/press/release.html

[2] "Zen+ - Microarchitectures - AMD." [Online]. Available: https://en.wikichip.org/wiki/amd/microarchitectures/zen%2B

[3] "Zen 3 - Microarchitectures - AMD." [Online]. Available: https://en.wikichip.org/wiki/amd/microarchitectures/zen_3

[4] "ECC Brings Reliability and Power Efficiency to Mobile Devices," Micron technology, Inc., Tech. Rep., 2017.

[5] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015. [Online]. Available: http://arxiv.org/abs/1508.03619

[6] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[8] S. Cha, O. Seongil, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. H. Ahn, and N. S. Kim, "Defect analysis and cost-effective resilience architecture for future dram devices," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 61–72.

[9] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving dram performance by parallelizing refreshes with accesses," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 356–367.

[10] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization," in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, ser. SIGMETRICS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 323–336. [Online]. Available: https://doi.org/10.1145/2896377.2901453

[11] J. Chung, I. Lee, M. Sullivan, J. H. Ryoo, D. W. Kim, D. H. Yoon, L. Kaplan, and M. Erez, "Containment domains: A scalable, efficient, and flexible resilience scheme for exascale systems," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

[12] I. Corporation, "8th and 9th Generation Intel Core Processor Families and Intel Xeon E Processor Families." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8th-gen-core-family-datasheet-vol-1.pdf

[13] S. Electronics, "Mobile DRAM Stack Specification."

[14] S.-L. Gong, J. Kim, and M. Erez, "Dram scaling error evaluation model using various retention time," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2017, pp. 177–183.

[15] S.-L. Gong, J. Kim, S. Lym, M. Sullivan, H. David, and M. Erez, "Duo: Exposing on-chip redundancy to rank-level ecc for high reliability," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 683–695.

[16] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[17] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*. New York: Springer SBM, 2011.

[18] M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[19] JEDEC, "DDR4 SDRAM Specication," 2012.

[20] JEDEC, "Low Power Double Data Rate 4 (LPDDR4) SDRAM Specication," 2014.

[21] S. Jeong, S. Kang, and J.-S. Yang, "Pair: Pin-aligned in-dram ecc architecture using expandability of reed-solomon code," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[22] M. Jung, C. Weis, N. Wehn, M. Sadri, and L. Benini, "Optimized active and power-down mode refresh control in 3d-drams," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, 2014, pp. 1–6.

[23] U. Kang, H. soo Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S.-J. Jang, and J. Choi, "Co-architecting controllers and dram to enhance dram process scaling," in *The Memory Forum*, 2014.

[24] J. Kim, M. Sullivan, and M. Erez, "Bamboo ecc: Strong, safe, and flexible codes for reliable computer memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 101–112.

[25] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 368–379.

[26] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 60–71. [Online]. Available: https://doi.org/10.1145/2485922.2485928

[27] I. Micron Technology, "8Gb: x4, x8, x16 DDR4 SDRAM."

[28] P. J. Nair, V. Sridharan, and M. K. Qureshi, "Xed: Exposing on-die error detection information for strong memory reliability," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 341–353.

[29] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 72–83. [Online]. Available: https://doi.org/10.1145/2485922.2485929

[30] Q. Nguyen, "RISC-V Tools (GNU Toolchain, ISA Simulator, Tests) – git commit 816a252." [Online]. Available: https://github.com/riscv/riscv-tools

[31] T.-Y. Oh, H. Chung, Y.-C. Cho, J.-W. Ryu, K. Lee, C. Lee, J.-I. Lee, H.-J. Kim, M. S. Jang, G.-H. Han, K. Kim, D. Moon, S. Bae, J.-Y. Park, K.-S. Ha, J. Lee, S.-Y. Doo, J.-B. Shin, C.-H. Shin, K. Oh, D. Hwang, T. Jang, C. Park, K. Park, J.-B. Lee, and J. S. Choi, "25.1 a 3.2gb/s/pin 8gb 1.0v lpddr4 sdram with integrated ecc engine for sub-1v dram core operation," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 430–431.

[32] S.-I. Pae, V. Kozhikkottu, D. Somasekar, W. Wu, S. G. Ramasubramanian, M. Dadual, H. Cho, and K.-W. Kwon, "Minimal aliasing single-error-correction codes for dram reliability improvement," *IEEE Access*, vol. 9, pp. 29 862–29 869, 2021.

[33] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and modeling on-die error correction in modern dram: An experimental study using real devices," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 13–25.

[34] M. Patel, J. S. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "Bit-exact ecc recovery (beer): Determining dram on-die ecc functions by exploiting dram data retention characteristics," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 282–297.

[35] Sanghyuk Kwon, Young Hoon Son, and Jung Ho Ahn, "Understanding ddr4 in pursuit of in-dram ecc," in *2014 International SoC Design Conference (ISOCC)*, 2014, pp. 276–277.

[36] Y. H. Son, S. Lee, O. Seongil, S. Kwon, N. S. Kim, and J. H. Ahn, "Cidra: A cache-inspired dram resilience architecture," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 502–513.

[37] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

[38] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014.

[39] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 175–186. [Online]. Available: https://doi.org/10.1145/1815961.1815983

[40] S. Wang, H. Hu, H. Zheng, and P. Gupta, "Memres: A fast memory system reliability simulator," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1783–1797, 2016.

[41] A. Waterman, "RISC-V Proxy Kernel – git commit 85ae17a." [Online]. Available: https://github.com/riscv/riscv-pk/commit/85ae17a

[42] A. Waterman and Y. Lee, “Spike, a RISC-V ISA Simulator – git commit 3bfc00e.” [Online]. Available: https://github.com/riscv/riscv-isa-sim

[43] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, “The RISC-V Instruction Set Manual Volume I: User-Level ISA Version 2.0,” 2014. [Online]. Available: https://riscv.org

[44] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, “AxBench: A Multiplatform Benchmark Suite for Approximate Computing,” *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2017.