Received 15 May 2024, accepted 15 July 2024, date of publication 19 July 2024, date of current version 29 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3431209

RESEARCH ARTICLE

ZEC ECC: A Zero-Byte Eliminating Compression-Based ECC Scheme for DRAM Reliability

JI HUN KWON[1], **HYEONG KON BAE**[1], **YOUNG SEO LEE**[2], **(Member, IEEE)**,
YOUNG-HO GONG[3], **(Member, IEEE)**,
AND SUNG WOO CHUNG[1], **(Senior Member, IEEE)**
[1]Department of Computer Science, Korea University, Seoul 02841, Republic of Korea
[2]School of Electronic Engineering, Soongsil University, Seoul 06978, Republic of Korea
[3]School of Software, Soongsil University, Seoul 06978, Republic of Korea

Corresponding authors: Young-Ho Gong (yhgong@ssu.ac.kr) and Sung Woo Chung (swchung@korea.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2C2003500, No. RS-2023-00212711, NRF-2022M3I7A2079155); in part by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00441-001, Memory-Centric Architecture Using the Reconfigurable PIM Devices); in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-RS-2022-00156225) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation); in part by Samsung Electronics Co., Ltd (project IO201212-08130-01); and in part by Korea University.

ABSTRACT As DRAM cells continue to shrink, the conventional single error correction and double error detection (SECDED) code is not sufficient to provide DRAM error resilience. To satisfy DRAM reliability demands, various studies have proposed multi-bit error correctable ECC schemes with substantial performance and/or storage overhead compared to the SECDED code. In this paper, we propose ZEC ECC, a zero-byte eliminating compression based ECC scheme, which provides much stronger error correction capability with negligible performance overhead and no storage overhead. ZEC ECC exploits our proposed Zero-byte Eliminating Compression (ZEC) to make room for additional parity bits. Depending on the compression ratio of a memory block ($\geq$60%, $\geq$50%, and <50%), ZEC ECC adaptively selects one out of three different ECC schemes (BCH(32,16,3), BCH(27,16,2), and BCH (573,512,6), respectively). Moreover, ZEC ECC tolerates a single chip failure by exploiting bitwise interleaving data placement, as long as the compression ratio is higher than or equal to 50% for a 64B memory block. Our experimental results show that ZEC ECC reduces the system failure probability (caused by DRAM errors) by 74.4%, on average, with only 1.6% performance overhead and no storage overhead compared to the conventional SECDED.

INDEX TERMS DRAM reliability, data compression, error correction code.

I. INTRODUCTION

With DRAM process technology scaling down, more DRAM cells are integrated in the same chip area, leading to higher DRAM density. On the other hand, the technology scaling makes DRAM cells more vulnerable to DRAM errors [18], [22] so that DRAM reliability has become a primary concern in data centers and supercomputers [9], [29]. To protect data from DRAM errors, error correction code (ECC) dual in-line memory modules (DIMMs) have been proposed to include a single error correction and double error detection (SECDED) code; the SECDED code is a widely-used ECC that corrects 1-bit error and detects 2-bit error per 64-bit data word by exploiting 8-bit parity. However, as DRAM error rate increases [9], [11], [14], [21], the conventional SECDED code is not sufficient to provide DRAM error resilience.

To satisfy memory reliability demands, several studies have proposed stronger ECC schemes than the conventional

The associate editor coordinating the review of this manuscript and approving it for publication was Lorenzo Ciani.

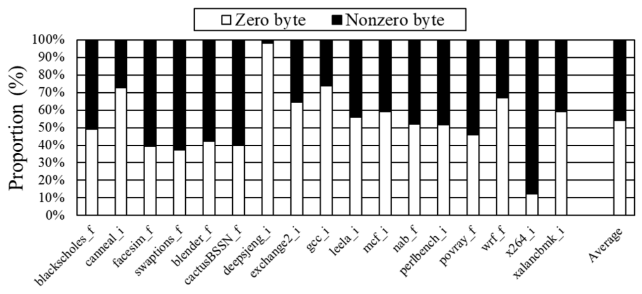


FIGURE 1. Proportion of zero/nonzero bytes in DRAM.

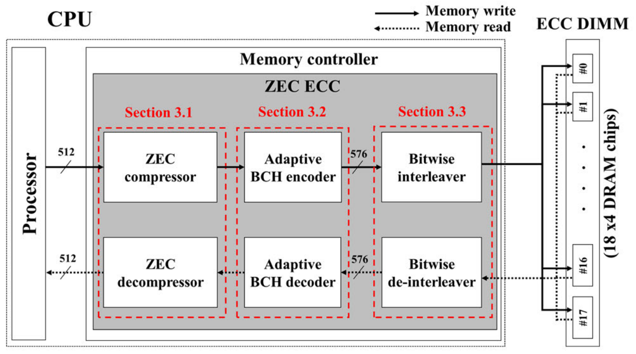


FIGURE 2. Overview of ZEC ECC.

SECDED code [9], [10], [13], [20], [25], [35]. A commercial Chipkill corrects any single chip failure and detects two chip failures by exploiting a symbol-based ECC with two channels of ECC DIMMs [10]. Though Chipkill can correct chip-level failures, it introduces performance overhead by up to 25% as it sacrifices bank/channel-level parallelism and incurs highest-priority interrupt for error correction [3], [9], [11]. To mitigate performance overhead, several studies proposed their ECC schemes providing Chipkill-level reliability. Frugal ECC delivers Chipkill-level reliability by employing data compression and a symbol-based ECC [20]. However, Frugal ECC causes storage overhead by up to 26%, since it exploits extra memory space to store parity bits in case of compression failure. LOT-ECC provides near Chipkill-level reliability by employing a multi-tier concept of separating error detection and error correction [35]. Similar to Frugal ECC, LOT-ECC incurs 26.5% storage overhead due to the parity bits for global error correction. CARE provides near Chipkill-level reliability by employing a 6-bit correctable ECC and page retirement schemes with operating system (OS) support [9], causing average 10% performance overhead due to the page retirement.

In this paper, we propose *ZEC ECC*, a compression-based adaptive ECC scheme, which provides stronger error correction capability than the conventional SECDED code with negligible performance overhead and no storage overhead. To store additional parity bits for stronger error correction, ZEC ECC compresses prevalent zero bytes (i.e., bytes whose values are zero) in a 64-byte (512-bit) memory block; note ZEC ECC *eliminates entire zero bytes,* achieving higher compression ratio compared to other compression methods [15], [19], [31]. Depending on the compression ratio (CR)[1] of each memory block, ZEC ECC employs different BCH (Bose-Chaudhuri-Hocquenghem) code to each memory block for maximizing error robustness, without any storage overhead. Even when a memory block could not be compressed by ZEC ECC, ZEC ECC does not degrade DRAM reliability by adopting SECDED to the memory block. In addition, ZEC ECC further enhances DRAM reliability through bitwise interleaving data placement across DRAM chips, providing single chip fault tolerance. In summary, leveraging data compression, ZEC ECC improves overall DRAM reliability with negligible performance overhead and no storage overhead.

II. BACKGROUND AND MOTIVATION

Bose-Chaudhuri-Hocquenghem (BCH) code is one of the most widely-used ECCs in DRAM, since it corrects and detects error(s) employing computation for parity check. Note BCH(n,k,t) encodes k-bit data into n-bit codeword by adding (n-k)-bit parity to provide t-bit error correction and (t+1)-bit error detection capability. To provide multi-bit error correction capability for 64-bit data word, it requires not only extra storage to store additional parity bits, but also additional performance overhead to decode BCH codes. To reduce such storage and/or performance overhead while providing stronger error correction capability, we deploy a novel data compression. Typically, it is well known that there are abundant zero bytes in DRAM [12], [15], [21]. We investigate the proportion of zero bytes in DRAM for 17 workloads from widely-used benchmark suites (SPEC CPU2017 [8] and PARSEC [5]); a zero byte indicates all the data bits in a byte are zeros (i.e., 00000000).

Figure 1 shows the proportion of zero bytes in DRAM. The experimental result shows that zero bytes account for 54.1% of all the bytes, on average. Especially, even in the case of floating point applications (denoted with _f), the proportion of zero bytes is not that small. Nevertheless, the previous compression methods are not effective to compress such prevalent zero bytes [15], [19], [31]. When eliminating zero bytes in DRAM, it is possible to free up space for additional parity bits and hence improve DRAM reliability without sacrificing available memory space. Based on this observation, we propose a data compression method, *Z̲ero-byte E̲liminating C̲ompression* (ZEC), which frees up space by eliminating entire zero bytes in a 64-byte memory block. Contrary to several ECC schemes relying on whether each 64-bit data is narrow (i.e., 32-bit consecutive zeros must be included in a 64-bit data) or not [2], [23], our proposed ZEC could compress all the zero bytes as long as the proportion of zero bytes are higher than 50% in a memory block, regardless of the stored patterns of zero bytes. Furthermore, leveraging

[1] $CR = \frac{Uncompressed\ data - Compressed\ data}{Uncompressed\ data} \times 100(\%)$.

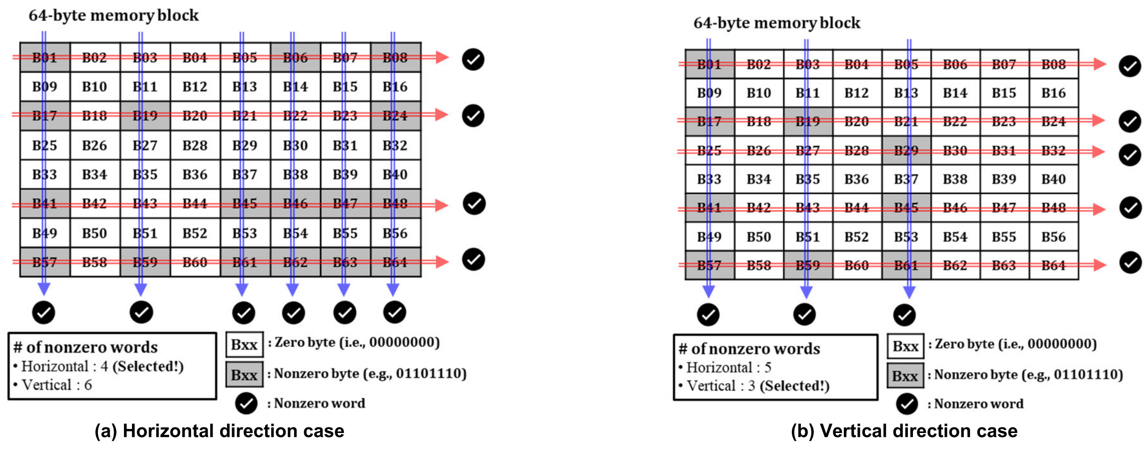

64-byte memory block
# of nonzero words
• Horizontal : 4 (Selected!)
• Vertical : 6
Bxx : Zero byte (i.e., 00000000)
Bxx : Nonzero byte (e.g., 01101110)
: Nonzero word
(a) Horizontal direction case
64-byte memory block
# of nonzero words
• Horizontal : 5
• Vertical : 3 (Selected!)
(b) Vertical direction case

FIGURE 3. The selection method of compression direction of ZEC in 64B memory block. A nonzero byte contains at least one nonzero (i.e., 1) bits.

our proposed ZEC, we propose a ZEC-based adaptive ECC scheme, *ZEC ECC*, which provides near Chipkill-level reliability without storage overhead compared to the conventional SECDED code.

III. ZEC (ZERO ELIMINATING COMPRESSION) ECC

Figure 2 shows the overall design of our proposed ZEC ECC encoding implemented in the memory controller. ZEC ECC encoding consists of three stages as follows. 1) In the compression stage, the ZEC compressor calculates the CR of a 64-byte memory block based on the proposed data compression method (i.e., ZEC). 2) In the adaptive ECC selection stage, the adaptive BCH encoder adopts an appropriate BCH code depending on the CR; when the CR is high, it is possible to adopt a stronger BCH code by storing more parity bits in the freed space. 3) In the bitwise interleaving stage, the encoded data bits are stored in an interleaved manner across 18×4 DRAM chips for reliability improvement. More details for three stages are described in the following subsections. In addition, we will explain the decoding procedure for ZEC ECC in Section III-D.

A. ZERO-BYTE ELIMINATING COMPRESSION

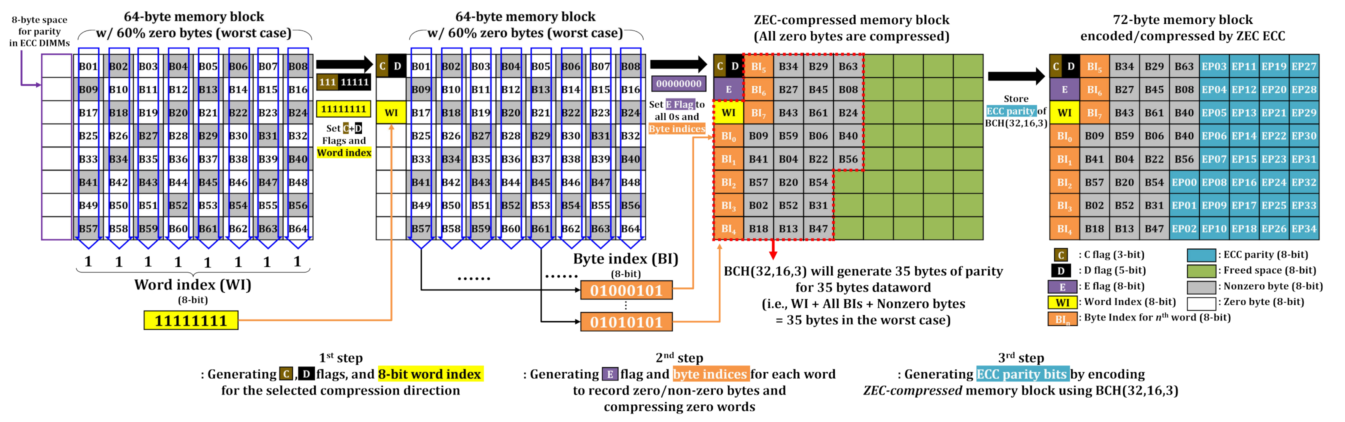
The ZEC compressor employs our proposed data compression method (i.e., ZEC) for a 64-byte memory block (which is same as L2 cache block size in this paper). To achieve high CR, ZEC considers both the horizontal and vertical directions for compression. Our approach is motivated by the ZERO compression [15]. The ZERO compression eliminates 8-byte zero words in the 64-byte memory block by considering both the horizontal and vertical directions; a zero word indicates a 8-byte data word where all the data bits are zeros, while a nonzero word indicates a 8-byte data word that has at least one data bit of '1'. As ZERO requires 64-bit consecutive zeros (i.e., zero word) for compression, it could not be effective when zero bytes are scattered in a memory block without a consecutive pattern. Due to the reason, there may still be

TABLE 1. Bits representataions of the C, D, E flags.

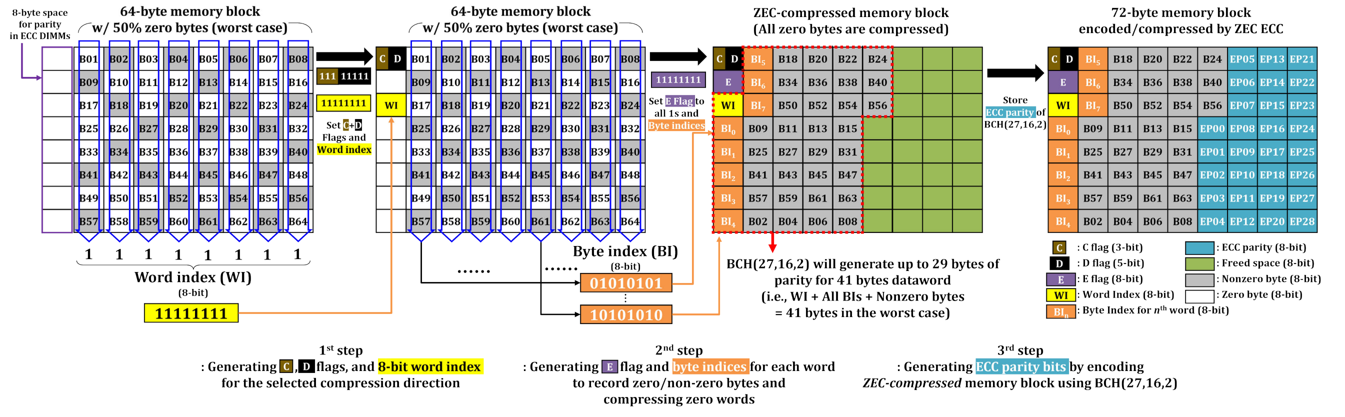
Flags (# bits)	Bit representation and description	
C flag (3b)	000 (Uncompressed)	111 (Compressed)
D flag (5b)	N/A	00000 (Horizontally compressed), 11111 (Vertically compressed)
E flag (8b)	N/A (BCH(573,512,6), if CR<50%)	00000000 (BCH(32,16,3), if CR≥60%), 11111111 (BCH(27,16,2), if CR≥50%)

many zero bytes in the ZERO-compressed memory block. Different from the ZERO compression, our proposed ZEC compresses the 64-byte memory block at a fine granularity (i.e., byte granularity), which in turn *fully eliminates zero bytes in nonzero words*.

To compress a memory block as small as possible (i.e., higher compression ratio), ZEC selects a compression direction where the memory block has smaller number of nonzero words. Thus, by counting the number of nonzero words in the horizontal/vertical direction, then selects the direction with smaller number of nonzero words as compression direction. Figure 3 illustrates the selection method of compression direction depending on the number of nonzero words in each direction; Figure 3 (a) and (b) show the horizontal and vertical direction case, respectively. As shown in Figure 3 (a), the memory block has four nonzero words in the horizontal direction. On the other hand, in the vertical direction, the memory block has six nonzero words. Since the number of nonzero words in the horizontal direction (i.e., four nonzero words) is smaller than that in the vertical direction (i.e., six nonzero words), ZEC selects the horizontal direction in this case, which makes more memory space for parity bits. For the vertical direction case shown in Figure 3 (b), since the number of nonzero words in the vertical direction is larger than that in the horizontal direction, the memory block would be compressed in the vertical direction.


8-byte space for parity in ECC DIMMs
64-byte memory block w/ 60% zero bytes (worst case)
Word index (WI) (8-bit)
11111111
Set C+D Flags and Word index
64-byte memory block w/ 60% zero bytes (worst case)
Byte index (BI) (8-bit)
01000101
01010101
Set E Flag to all 0s and Byte indices
ZEC-compressed memory block (All zero bytes are compressed)
BCH(32,16,3) will generate 35 bytes of parity for 35 bytes dataword (i.e., WI + All BIs + Nonzero bytes = 35 bytes in the worst case)
Store ECC parity of BCH(32,16,3)
72-byte memory block encoded/compressed by ZEC ECC
C : C flag (3-bit)
D : D flag (5-bit)
E : E flag (8-bit)
WI : Word Index (8-bit)
BI_n : Byte Index for nth word (8-bit)
: ECC parity (8-bit)
: Freed space (8-bit)
: Nonzero byte (8-bit)
: Zero byte (8-bit)
1st step : Generating C, D flags, and 8-bit word index for the selected compression direction
2nd step : Generating E flag and byte indices for each word to record zero/non-zero bytes and compressing zero words
3rd step : Generating ECC parity bits by encoding ZEC-compressed memory block using BCH(32,16,3)

(a) Encoding process for a memory block w/ CR≥60%. Note the figure depicts encoding of a memory block w/ CR=60% using BCH(32,16,3).


8-byte space for parity in ECC DIMMs
64-byte memory block w/ 50% zero bytes (worst case)
Word index (WI) (8-bit)
11111111
Set C+D Flags and Word index
64-byte memory block w/ 50% zero bytes (worst case)
Byte index (BI) (8-bit)
01010101
10101010
Set E Flag to all 1s and Byte indices
ZEC-compressed memory block (All zero bytes are compressed)
BCH(27,16,2) will generate up to 29 bytes of parity for 41 bytes dataword (i.e., WI + All BIs + Nonzero bytes = 41 bytes in the worst case)
Store ECC parity of BCH(27,16,2)
72-byte memory block encoded/compressed by ZEC ECC
C : C flag (3-bit)
D : D flag (5-bit)
E : E flag (8-bit)
WI : Word Index (8-bit)
BI_n : Byte Index for nth word (8-bit)
: ECC parity (8-bit)
: Freed space (8-bit)
: Nonzero byte (8-bit)
: Zero byte (8-bit)
1st step : Generating C, D flags, and 8-bit word index for the selected compression direction
2nd step : Generating E flag and byte indices for each word to record zero/non-zero bytes and compressing zero words
3rd step : Generating ECC parity bits by encoding ZEC-compressed memory block using BCH(27,16,2)

(b) Encoding process for a memory block w/ 50%≤CR<60%. Note the figure depicts encoding of a memory block w/ CR=50% using BCH(27,16,2).

FIGURE 4. ZEC ECC encoding process of a compressible memory block depending on the compression ratio.

If both directions have the same number of nonzero words, any direction could be selected for compression, which is implementation-dependent.

Depending on the proportion of zero bytes in a memory block, ZEC stores 3-bit *C (Compression) flag* to a memory block, to identify whether the memory block is compressed (111) or not (000); using 3-bit redundancy for C flag makes C flag tolerable to single-bit error. Also, if the memory block is compressed (i.e., *C flag* = 111), ZEC stores the *D (direction) flag*to identify the compression direction; the *D flag* of all 0s (or all 1s) indicates that the memory block is compressed horizontally (or vertically).

B. ADAPTIVE ECC SELECTION

To provide adaptive ECC depending on the CR of a 64-byte memory block, there needs to be a flag to classify the CR. Hence, in our ZEC ECC, the adaptive BCH encoder generates *E (ECC) flag as well as C and D flags* for a compressible memory block, depending on the CR. Table 1 summarizes the information and bit representations of *C, D, and E Flags*. The *E flag* represents which BCH code is used for ECC depending on the CR; when CR≥60%, BCH(32,16,3) is applied to the compressed memory block and word/byte indices. Otherwise (i.e., 50%≤CR<60%), BCH(27,16,2) is applied. To avoid additional storage overhead for flags, we store the *C, D, and E flags* in the freed space of the compressed memory block when CR≥50%. Though we denote *C and D flags* separately in Table 1, they are stored in the first single byte (8-bit) of a compressible memory block. Meanwhile, when a memory block is identified as non-compressible (CR<50%), instead of adopting the conventional SECDED code (BCH(72,64,1)), we adopt BCH(573,512,6) to obtain free (3-bit) space in the 576-bit (corresponding to eight memory bursts). By exploiting the 3-bit freed space in an uncompressed memory block, *C flag* is stored as 000 (as described in Table 1) to represent the memory block is uncompressed.

Figure 4 (a) and (b) show the encoding process of a compressible memory block with our proposed ZEC ECC, when the memory block could be compressible at 50% and 60% rate, respectively. Note Figure 4 (a) and (b) represent the *worst-case* scenarios (i.e., checkerboard word pattern where all words are non-zero words but including zero bytes) for each case. We explain how our proposed ZEC ECC encodes the 60% compressible memory block. First, as shown in the leftmost of Figure 4 (a), ZEC selects the compression direction, depending on the number of nonzero words for

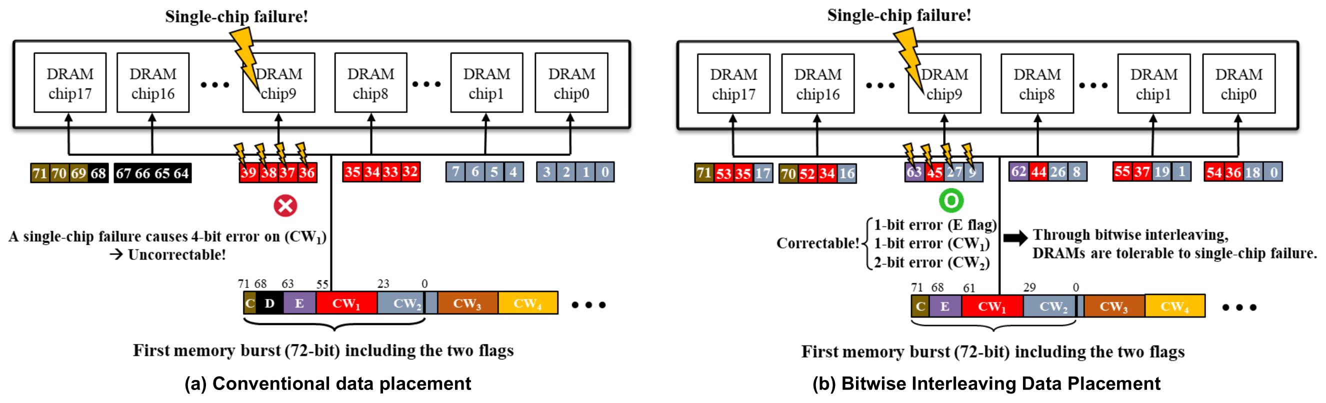

Single-chip failure!
DRAM chip17
DRAM chip16
DRAM chip9
DRAM chip8
DRAM chip1
DRAM chip0
71 70 69 68
67 66 65 64
39 38 37 36
35 34 33 32
7 6 5 4
3 2 1 0
A single-chip failure causes 4-bit error on (CW1) → Uncorrectable!
C D E CW1 CW2 CW3 CW4
First memory burst (72-bit) including the two flags
(a) Conventional data placement
71 53 35 17
70 52 34 16
63 45 27 9
62 44 26 8
55 37 19 1
54 36 18 0
Correctable!
1-bit error (E flag)
1-bit error (CW1)
2-bit error (CW2)
Through bitwise interleaving, DRAMs are tolerable to single-chip failure.
C E CW1 CW2 CW3 CW4
First memory burst (72-bit) including the two flags
(b) Bitwise Interleaving Data Placement

FIGURE 5. Bit mapping depending on data placement methods in the first memory burst of the 64-byte memory block encoded by BCH(32,16,3). (CW=codeword).

each direction. When ZEC selects the compression direction, it generates *word index*for the memory block. In the *word index*, the bit of 0 (or 1) indicates a zero word (or a nonzero word). Since the memory block shown in Figure 4 does not have zero words, ZEC sets all 1s for the 8-bit word index (WI). Also, ZEC sets all 1s for the *C*and *D flags*, to indicate the memory block is vertically compressed; if the memory block is compressed in a horizontal direction, the *D flag* is set to all 0s. Note, by employing 5-bit redundancy where all five bits are either all 0s or 1s, the *D flag*is able to tolerate up to 2-bit error; a compressed memory block can be decompressed by counting the number of 0s in the direction bits. Second, ZEC records all the positions of zero/nonzero bytes in the *byte index*. Thus, it is possible to *eliminate entire zero bytes in a memory block* without data loss, thereby resulting in high CR. In *byte index*, the bit of 0 (or 1) indicates a zero byte (or a nonzero byte). In the worst case scenario of 60% compressible memory block (when all words include any single nonzero byte, e.g., checkerboard pattern) ZEC requires 72 bits for word/byte indices (i.e., 8-bit for word index + 8-bit * 8 words for byte indices).

After a memory block is compressed by ZEC, there would be enough free space to store ECC parity bits. Thus, ZEC ECC encodes the memory block with the corresponding BCH code. The BCH code is applied to the *word/byte indices* as well as the compressed memory block. As shown in Figure 4 (a), in case of CR≥60%, up to 35 bytes (i.e., 9 bytes for word/byte indices + 26 bytes for data) are encoded by BCH(32,16,3), which in turn generates 35 bytes of parity. As a result, 70 bytes (i.e., 9 bytes for word/byte indices + 26 bytes for data + 35 bytes for parity) are stored with 2 bytes of flags, which are totally fit into a 72-byte memory block.

If the data size to be encoded is larger than 35 bytes (i.e., CR<60%), it cannot be encoded with BCH(32,16,3). When the data is compressed with 50%≤CR<60%, ZEC ECC applies BCH(27,16,2). Figure 4 (b) shows the worst-case scenario of encoding memory block with CR=50%, where BCH(27,16,2) is applied. As shown in Figure 4 (b), in case of CR=50%, 41 bytes (i.e., 9 bytes for word/byte indices + 32 bytes for data) are encoded by BCH(27,16,2), generating 29 bytes of parity. To summarize, in case of CR=50%, 72 bytes are required to store flags, indices, data, and parity for the 50% compressible memory block, which are also fit into 72-byte memory block.

C. BITWISE INTERLEAVING FOR SINGLE CHIP TOLERANCE

After encoding a memory block using ZEC ECC, the bitwise interleaver reorganizes the codeword and flag bits by exploiting the bitwise interleaving data placement, to tolerate chip failure [2], [23]. Figure 5 describes the bit mapping of ZEC ECC with the conventional and bitwise interleaving data placement, when storing the first memory burst (72-bit) of the 64-byte memory block encoded by BCH(32,16,3). As shown in Figure 5(a), in case of the conventional data placement, a single memory burst (72-bit) is divided into 18 fractions in the unit of 4 sequential bits and then each fraction (i.e., sequential 4 bits) is stored in a single DRAM chip. In this case, since a single chip failure results in up to 4-bit errors in a codeword, ZEC ECC would not be robust to a single chip failure; ZEC ECC employs up to 3-bit correctable BCH code when CR≥60%. On the other hand, as shown in Figure 5(b), in case of the bitwise interleaving data placement, the sequential data bits are stored in an interleaved manner across 18 × 4 DRAM chips. In this case, since a single chip failure contains up to 2-bit errors in a codeword, the BCH(32,16,3) and BCH(27,16,2) codes in ZEC ECC are capable of correcting a single chip failure. In addition, a single chip failure contains up to 1-bit error in the flag bits, which is tolerable by the redundant bits of the flags (as explained in Section III-B). Thus, with the BCH(32,16,3) and BCH(27,16,2) codes (when CR≥50%), ZEC ECC is robust to a single chip failure through bitwise interleaving data placement, which in turn improves overall DRAM reliability.

D. DECODING PROCESS OF ZEC ECC FOR READ OPERATIONS

We describe how ZEC ECC compresses/encodes a memory block depending on the CR in Sections III-A–III-C.

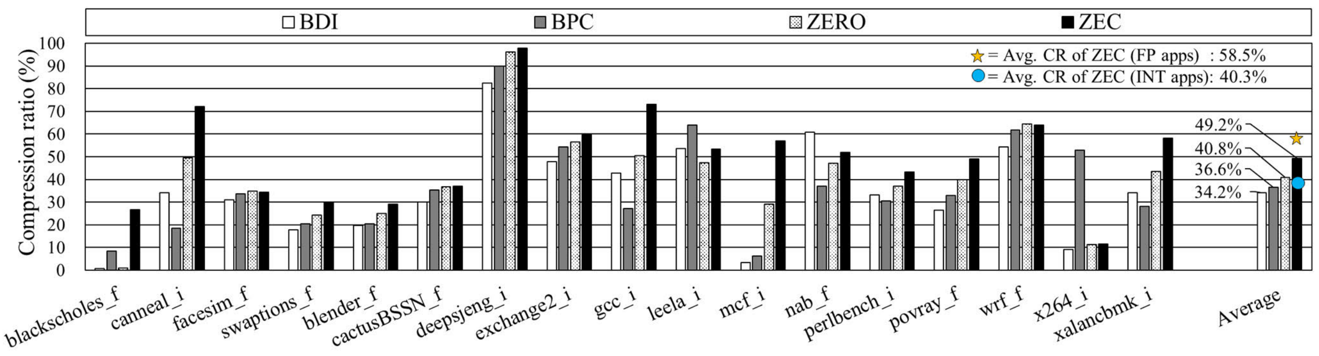


FIGURE 6. Compression ratio depending on compression methods (The higher, the better).

To support read operations of the memory block encoded by ZEC ECC, our proposed design requires bitwise de-interleaver, adaptive BCH decoder, and ZEC decompressor in the memory controller, as shown in Figure 2. First, when reading a 72-byte memory block from DRAM chips, the bitwise de-interleaver reorganizes the interleaved encoded data bits (i.e., codeword(s) and flag bits). Then, as shown in Figure 2, the adaptive BCH decoder identifies 1) whether the memory block is compressed or not (by reading *C flag*), 2) which direction is used for compression (by reading *D flag*), and 3) which BCH code needs to be used for decoding the memory block (by reading *E flag*), by counting the number of zeros in the *C, D, and E flags*; note ZEC ECC redundantly stores the *C, D, and E flags*to tolerate the error(s) within the flags. Since the flags are included in the first memory burst, counting number of zeros in each flag can be done before the remaining memory bursts arrive. Therefore, the latency to count zeros in the flags will be hidden to the memory read latency. Similarly, the word/byte indices are included in the second memory burst (even in the worst-case scenario). Thus, the latency of ZEC decompressor can be overlapped to the latency until all the memory bursts arrive. Once all the memory bursts have been read from DRAM chips, the memory block is decoded by the adaptive BCH decoder, depending on the *E flag*. The ZEC decompressor then begins to decompress the decoded memory block based on the word/byte indices. Note our decompression is not complicated so that it incurs negligible latency overhead.

IV. EVALUATION

A. EVALUATION SETUP

We evaluate ZEC ECC in terms of compression ratio/coverage, reliability, performance, and area/power overhead. We conduct our evaluations on 17 workloads, from SPEC CPU 2017 [8] and PARSEC [5] benchmark suites. For the compression ratio/coverage,[2] to conservatively compare our proposed ZEC with the other compression schemes, we executed 11 billion instructions and then captured the memory snapshot; if we execute only a few or hundreds of million instructions, most of the malloc'd data may not be physically written to DRAM due to the caching effect. In this case, most memory blocks in DRAM still have zeros, while they are malloc'd, which is too advantageous for ZEC. To minimize the caching effect, we consider a sufficiently large number of instructions, so that the malloc'd memory blocks have non-zero data as much as possible [2], [23]. Based on this environment, we calculated the compression ratio/coverage for only malloc'd regions, not including unused memory spaces as zeros.

[2]In this paper, the compression coverage is defined as the proportion of compressible (i.e., CR=50%) memory blocks to all memory blocks.

TABLE 2. Configuration for performance simulation.

Parameter	Configuration
Processor	x86; 7-stage out-of-order pipelined; 3GHz
Cache	L1D/L1I: 32KB; 8-way; 64B block L2: 256KB; 4-way; 64B block
Memory controller	ECC decoding latency (cycle): Baseline* (3), BCH(27,16,2) (6), BCH(32,16,3) (9), BCH(573,512,6) (24), Stealth ECC (9), Twin ECC (7) Decompression latency: 1 cycle
Main memory	DDR4-2400; x4 bus-width; 18 Chips; 16GB

*Note our baseline is BCH(72,64,1).

For the reliability evaluation, we exploit FaultSim [27], a memory-reliability simulator, which employs Monte Carlo simulations based on real-world DRAM failure statistics. To evaluate the system failure probability in 7 years, we perform 1 million Monte Carlo trials for each ECC scheme. For the performance evaluation, we modify gem5 [6] to reflect the ECC decoding and decompression latency. We calculate the ECC decoding latency of each ECC scheme exploiting Strukov's model [34]; the decoding latency for each ECC scheme is described in Table 2. The baseline of our evaluation is the conventional SECDED (i.e., BCH(72,64,1)). In case of the SPEC CPU 2017 and PARSEC workloads, each workload is fast-forwarded 10 billion instructions and then executed one billion instructions. Table 2 provides the detailed system configuration for the performance simulation. For the area/power overhead of ZEC ECC, we implement the additional hardware components of ZEC ECC in Verilog HDL and then synthesize them using Synopsys Design Compiler based on SAED 14nm FinFET process technology [26]. In addition to the ECC decoding latency as described in Table 2, we reflect one additional cycle of the decompression latency based on our synthesis/analysis result; the decompression latency of ZEC is 0.32 ns. Note it includes the latency to access 1KB SRAM read buffer with 64-bit XOR gates for checking compression direction by counting zero/non-zero words, based on FinCACTI [33] with 14nm FinFET technology.

TABLE 3. Comparison of compression methods.

	Description
BDI [31]	• Compresses a cacheline by exploiting the base and differences. • Compression granularity: 32-byte, 64-byte*
BPC [19]	• Compresses a 128-byte data by exploiting Delta-BitPlane-XOR (DBX) data transformation. • Compression granularity: 128-byte*
ZERO [15]	• Compresses a 64-byte memory block by minimizing the number of nonzero words (*but still remaining a number of zero bytes in nonzero words*). • Compression granularity: 64-byte

*For fair comparison, we set the compression granularities of BDI and BPC to 64-byte and then measure the compression ratio/coverage of BDI and BPC.

B. COMPRESSION RATIO/COVERAGE

Figure 6 shows the compression ratio of BDI [31], BPC [19] (these two methods were originally proposed not for reliability but for compression itself), ZERO [15], and ZEC; BDI, BPC, and ZERO are briefly explained in Table 3. Note BDI and BPC are still widely considered in recent studies [17], [24], [32], as they have a good tradeoff between hardware cost and compressibility. We consider ZERO as a competitive to ZEC, since ZERO is also a zero compression technique for ECC. As shown in Figure 6, the average compression ratio of ZEC is 48.3% which is the highest average compression ratio among four different compression methods. For example, in the case of *mcf*, while BDI, BPC, and ZERO shows 3.4%, 6.2%, and 29.0% average compression ratio, respectively, ZEC achieves 48.8% average compression ratio. Though BDI and BPC show higher CR than ZEC in a few applications (e.g., *leela*), they are only effective for a memory block with a very-limited range value pattern. Thus, ZEC shows much higher CR than BDI and BPC on average. As discussed in Section III-A, ZEC also outperforms ZERO on average, since it can compress zero bytes even in nonzero words. More importantly, in case of *blackscholes* which is the most hard-to-compress application, though the other compression methods are not effective (CR$<$10%), ZEC achieves 26.7% CR, thanks to the byte-wise zero compression.

Figure 7 shows the compression coverage[2] depending on the compression methods. As shown in Figure 7, ZEC shows the highest average compression coverage (63.7%) among four different compression methods, since it eliminates all the zero bytes for compressible memory blocks, considering both horizontal and vertical words. In addition, ZEC provides much higher compression coverage than the other compression methods, for some hard-to-compress applications like *blackscholes*and *mcf*. Hence, it is possible for ZEC to make more space for additional parity bits, compared to other compression methods.

C. RELIABILITY

Figure 8 shows the system failure probability depending on the ECC schemes. ZEC ECC reduces the average system failure probability by 74.4%, 63.9%, 49.0%, and 23.7% compared to the baseline, BCH(573,512,6), Stealth ECC [23], and Twin ECC [2], respectively.

The system failure probability of ZEC ECC relies on the compression coverages of ZEC. As the compression coverage increases, ZEC ECC significantly reduces the system failure probability by exploiting the stronger BCH codes. For example, as shown in Figure 8, the system failure probability with ZEC ECC becomes extremely low in case of *canneal*, *deepsjeng*, *gcc*, *mcf*, *nab*, and *xalancbmk*, since the compression coverage of ZEC is extremely high in those applications.

D. PERFORMANCE

Similar to the previous studies [15], [23], we assume that the compression and ECC encoding process are performed in a pipelined manner when the evicted block from the last level cache (LLC) is waiting on the write queue of the memory controller. Thus, in ZEC ECC, the compression and ECC encoding latency do not degrade system performance.

On the other hand, the decompression and ECC decoding latency may degrade the system performance, since these latencies lie on the critical memory access path. However, as discussed in Section III-D, the decoding latency of flag and word/byte indices (except BCH decoding latency) could be hidden to the read latency, while we reflect one additional cycle for the decompression latency. Figure 9 shows the normalized execution time depending on the ECC schemes. ZEC ECC shows the average performance overhead by 1.6% compared to the baseline. In ZEC ECC, as the applied BCH code differs depending on the CR, the total latency (= decompression + ECC decoding latency) also varies among 7, 10, and 24 cycles. Though the longest latency of ZEC ECC is 24 cycles, the performance overhead of ZEC ECC is not that significant due to the adaptive ECC selection. For example, in case of *blackscholes*, the most hard-to-compress application with only 26.7% compression ratio, the performance overhead is only 2.4% compared to the baseline. In addition, in case of *wrf* (the worst-case execution time), though the application tends to access non-compressed memory blocks frequently, the performance overhead is not so significant (4.3%).

Note, in case of CR$\geq$50%, the total latency of ZEC ECC is 7 (BCH(27,16,2)) or 10 (BCH(32,16,3)) cycles. Since the average compression coverage of ZEC is 63.7% as described in Figure 7, it is not so frequent to suffer the longer decoding latency (24 cycles in case of CR$<$50%). Stealth ECC and

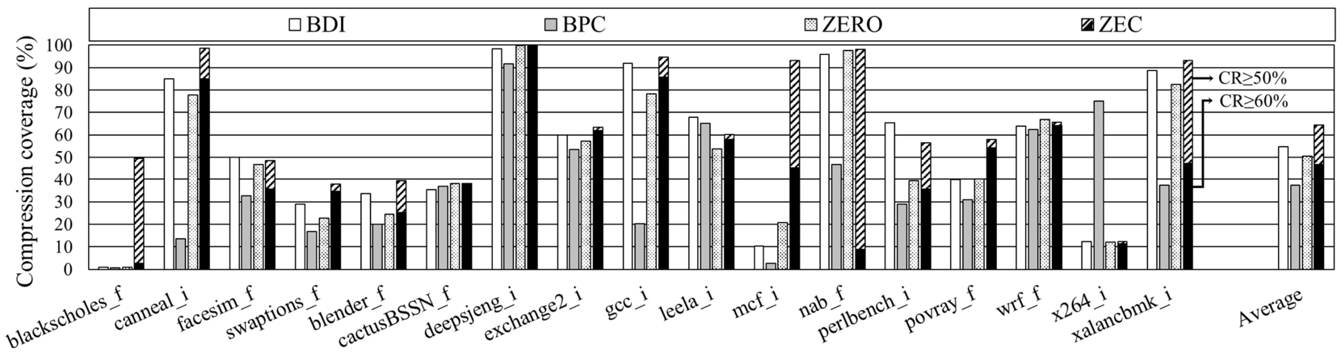


FIGURE 7. Compression coverage depending on compression methods (The higher, the better).

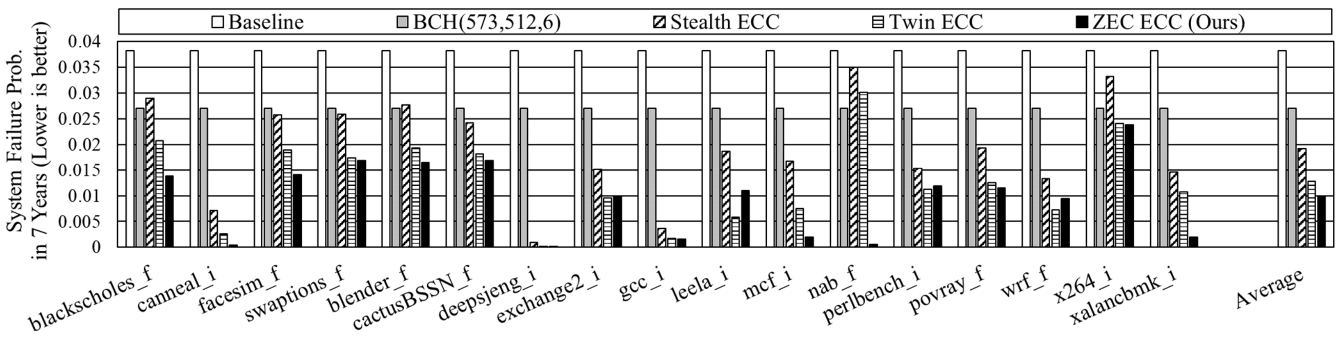


FIGURE 8. System failure probability in 7 years (The lower, the better).

Twin ECC show the average performance overhead by 0.9% and 0.6%, compared to the baseline, respectively, which are slightly lower than ZEC ECC. However, with negligible performance overhead, ZEC ECC reduces the average system failure probability by 49.0% and 23.7% compared to Stealth ECC and Twin ECC, respectively.

E. AREA/POWER OVERHEAD

Our synthesis result shows that ZEC ECC occupies 0.24 mm^2 and consumes 10.16 mW. Considering that the memory controller of the state-of-the-art server CPU [36] is about 6.9 mm^2, ZEC ECC incurs negligible area overhead (i.e., 3.2% of the memory controller area). In addition, compared to the die sizes of the state-of-the-art server CPUs [36], ZEC ECC incurs an area overhead by only 0.06%. The power overhead (10.16mW) of ZEC ECC is also negligible compared to the thermal design power (i.e., 205W) of the state-of-the-art server CPU. Due to the negligible power/area overhead, even if we consider an additional redundant ZEC ECC logic for logic error robustness, it does not incur much power/area overhead.

F. SENSITIVITY TO COMPRESSION COVERAGE

We investigate the relation between compression coverage and reliability improvement of ZEC ECC. Figure 10 shows the system failure probability of four different ECC schemes with the different compression coverage. To conservatively evaluate the reliability improvement, we assume that ZEC ECC adopts BCH(27,16,2) in case of CR$\geq$50%.

As the compression coverage increases, ZEC ECC reduces system failure probability. As shown in Figure 10, when the compression coverage is higher than 89.3%, ZEC ECC provides lower system failure probability than Chipkill. Even in a case that the compression coverage is 0%, ZEC ECC shows lower system failure probability compared to the baseline, since it adopts BCH(573,512,6) for non-compressible memory blocks (i.e., CR$<$50%).

V. RELATED WORK

Many previous studies have presented ECC schemes to protect main memory. Among them, we describe the previous studies that provided moderate level or Chipkill-level error protection, as shown in Table 4; note our simulation environments described in Section IV are similar to those for the Chipkill-level ECC schemes. The three leftmost studies [15], [16], [30] in Table 4 have presented compression-based ECC schemes. However, they have only 1-bit or 2-bit error(s) correction capability and do not provide Chipkill-level protection.

For DRAM systems requiring high reliability, the rest of the studies [2], [9], [14], [18], [19], [23], [35] in Table 4 have

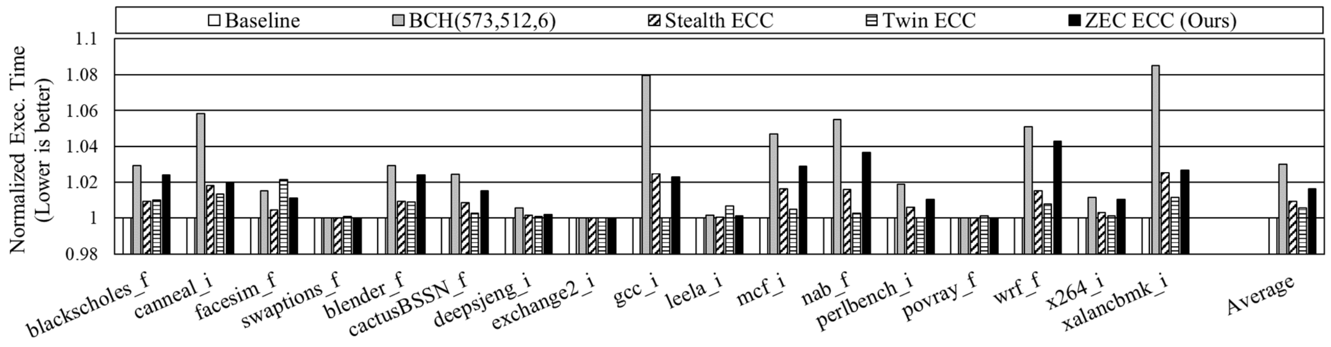


FIGURE 9. Normalized execution time (The lower, the better).

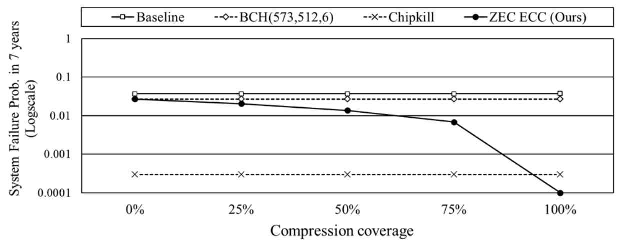


FIGURE 10. System failure probability depending on the compression coverage).

presented Chipkill-level ECC schemes. Gong et al. [14] and Kim et al. [18], [19] proposed ECC schemes which apply the symbol-based linear block code. Udipi et al. proposed LOT-ECC [35] which exploits a multi-tier concept of separating EDC (error detection code) and ECC. Chen et al. presented CARE [9], which exploits a 6-bit error correction ECC scheme with operating system (OS) support. Though the above mentioned studies tolerate a single chip failure, most of them cause significant performance and/or storage overhead compared to the conventional SECDED code. Different from the above studies, ZEC ECC tolerates a single chip failure through compression with 1) no storage overhead compared to the conventional SECDED code, 2) negligible performance overhead, 3) no OS support, and 4) no LLC modification.

In the meantime, to reduce performance and storage overhead, Lee et al. [23] proposed Stealth ECC, a data-width aware adaptive ECC scheme, which provides near Chipkill-level reliability by adaptively selecting BCH codes depending on the data-width (either narrow-width or full-width). Though Stealth ECC applies a stronger BCH code to a narrow-width value without additional storage overhead, there are still wasted space to store zero bytes in narrow-width values, which is resolved in ZEC ECC. In addition, Bae et al. proposed Twin ECC [2] a duplication-based ECC scheme providing strong reliability for specific data patterns. Though Twin ECC delivers stronger error correction with duplication than Stealth ECC, its coverage is not so high in real applications, as it requires specific data patterns within each single memory block. On the other hand, ZEC ECC covers much more memory blocks than Twin ECC, since it leverages a fine grain compression for all the zero bytes even in irregular data patterns, which in turn provides better reliability.

VI. DISCUSSON

A. COMPRESSION/DECOMPRESSION FAILURE HANDLING

Since ZEC ECC is based on the compression and decompression of a memory block, we need to consider the failure of compression/decompression. In the case of compression failure, which means that a memory block cannot be compressed with CR$\geq$50% by ZEC, our scheme just stores the original data by applying BCH(573,512,6). Thus, it is not a problem for normal operations.

However, in the case of decompression failure, it may result in system failure. The decompression can fail due to 1) the uncorrectable error on the memory block or 2) decompression logic error. First, in the case of uncorrectable error-induced failure, since ZEC ECC has much higher error correctability than the baseline DRAM ECC, the decompression failure from uncorrectable error would be negligible. Note ZEC ECC leads to 74.4% reduction in system failure probability as shown in Figure 8, compared to the baseline SECDED. Second, in the case of decompression logic error, since the logic error rate is much lower than DRAM error rate [7], it would not be a major concern. Nevertheless, the decompression logic error can be mitigated significantly when using redundant decompression logic. The power/area overhead of redundant decompression logic would be negligible, since our decompression logic has extremely smaller power/area overhead, compared to the memory controller logic, as described in Section IV-E.

B. IMPACT ON APPLICATIONS WITH LOW DATA LOCALITY

When an application has low cache hit rate (i.e., low data locality), ZEC ECC may cause performance overhead due to frequent memory accesses to DRAM, leading to memory block decompression overhead; note the compression latency does not degrade the performance as we described in Section IV-D. However, according to our analysis, ZEC ECC has only a small performance impact on applications with low data locality. For example, among the benchmark

TABLE 4. Summary of related work and ZEC ECC.

Analysis point	Moderate level ECC schemes			Chipkill-level ECC schemes								
	EAR [15]	COP [30]	LoComp [16]	CLEAN-ECC [14]	Bamboo ECC [18]	FECC+ OPC [19]	FECC+ Multi [19]	LOT-ECC [35]	CARE [9]	Stealth ECC [23]	Twin ECC [2]	**ZEC ECC (This work)**
Modified components*	MC	OS, MC	MC	LLC, MC	MC	MC	MC	MC	OS, MC	MC	MC	MC
Storage cost (vs. non-ECC DIMM)	0.85%	9.38%	3.1%	12.5%	12.5%	26%	13.5%	26.5%	12.5%	12.5%	12.5%	12.5%
Perf. overhead (vs. SECDED⁴)	0.9%	1.3%	1%	19.3%	20.3%	21.4%	21.7%	13.3%	10%¹	0.9%	0.6%	1.6%
Reliability level	≤DECTED²	SECDED³	SECDED⁴	Chipkill	Chipkill	Chipkill	≤Chipkill	≤Chipkill	≤Chipkill	≤Chipkill†	≤Chipkill†	≤Chipkill†

*Abbreviation: MC-Memory Controller, OS-Operating System, LLC-Last Level Cache.
¹Page retirement overhead, ²BCH(79,64,2) code, ³BCH(128,120,1) code, ⁴BCH(72,64,1) code
† ZEC ECC improves DRAM reliability by 49.0% and 23.7% than Stealth ECC and Twin ECC, respectively, as described in Section IV-C.

applications, *canneal_i* and *mcf_i*are the representative applications with high LLC MPKI (Misses Per Kilo Instructions) [4], [28]. According to our simulation on *canneal_i* and *mcf_i*, ZEC ECC incurs only 1.9% and 2.8% performance overhead, respectively, compared to the baseline, while mitigating 98.8% and 94.9% system failure probability. Though we do not include cache prefetching in our simulation, the performance overhead on applications with low data locality can be mitigated when ZEC ECC is applied with cache prefetching techniques [1].

VII. CONCLUSION

We propose ZEC ECC, a ZEC-based adaptive ECC scheme that compresses a 64-byte memory block and adaptively selects BCH codes depending on the compression ratio of each memory block. ZEC ECC delivers up to Chipkill-level reliability by tolerating a single chip failure in case of CR$\geq$50%. Consequently, ZEC ECC reduces the average system failure probability (caused by DRAM errors) by 74.4% with negligible performance overhead (1.6%, on average) and without any storage overhead, compared to the conventional SECDED code. Moreover, ZEC ECC is more reliable than Stealth ECC (by 49.0%) and Twin ECC (by 23.7%), which are the two most reliable schemes with negligible performance overhead and without storage overhead. We expect that ZEC ECC will be applicable to other memory technologies including non-volatile memories such as ReRAM, FeRAM, MRAM, providing significant reliability improvement with negligible overhead.

REFERENCES

[1] G. Ayers, H. Litz, C. Kozyrakis, and P. Ranganathan, ‘‘Classifying memory access patterns for prefetching,’’ in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2020, pp. 1–26.

[2] H. K. Bae, M. J. Chung, Y.-H. Gong, and S. W. Chung, ‘‘Twin ECC: A data duplication based ECC for strong DRAM error resilience,’’ in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, Apr. 2023, pp. 1–20.

[3] M. V. Beigi, S. Gurumurthi, and V. Sridharan, ‘‘Reliability, availability, and serviceability challenges for heterogeneous system design,’’ in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Mar. 2022, pp. 2C41–2C48.

[4] M. Bhadauria, V. M. Weaver, and S. A. McKee, ‘‘Understanding PARSEC performance on contemporary CMPs,’’ in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 98–107.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, ‘‘The PARSEC benchmark suite: Characterization and architectural implications,’’ in *Proc. Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Oct. 2008, pp. 72–81.

[6] 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,’’ *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–8, 2011.

[7] L. Borucki, G. Schindlbeck, and C. Slayman, ‘‘Comparison of accelerated DRAM soft error rates measured at component and system level,’’ in *Proc. IEEE Int. Rel. Phys. Symp.*, Apr. 2008, pp. 1–18.

[8] J. Bucek, K.-D. Lange, and J. v. Kistowski, ‘‘SPEC CPU2017: Next-generation compute benchmark,’’ in *Proc. Companion ACM/SPEC Int. Conf. Perform. Eng.*, Apr. 2018, pp. 1–28.

[9] J. Chen, X. Jiang, Y. Zhang, L. Liu, H. Xu, and Q. Liu, ‘‘CARE: Coordinated augmentation for elastic resilience on DRAM errors in data centers,’’ in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 533–544.

[10] T. J. Dell, ‘‘A white paper on the benefits of chipkill-correct ECC for PC server main memory,’’ *IBM Microelectron. Division*, vol. 11, pp. 1–23, Jul. 1997.

[11] X. Du and C. Li, ‘‘DPCLS: Improving partial cache line sparing with dynamics for memory error prevention,’’ in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 197–204.

[12] M. Ekman and P. Stenstrom, ‘‘A robust main-memory compression scheme,’’ in *Proc. 32nd Int. Symp. Comput. Archit.*, 2005, pp. 1–29.

[13] Y. Fang, G. Han, G. Cai, F. C. M. Lau, P. Chen, and Y. L. Guan, ‘‘Design guidelines of low-density parity-check codes for magnetic recording systems,’’ *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1574–1606, 2nd Quart., 2018.

[14] S.-L. Gong, M. Rhu, J. Kim, J. Chung, and M. Erez, ‘‘CLEAN-ECC: High reliability ECC for adaptive granularity memory system,’’ in *Proc. 48th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2015, pp. 611–622.

[15] J. Hong, H. Kim, and S. Kim, ‘‘EAR: ECC-aided refresh reduction through 2-D zero compression,’’ in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn.*, Nov. 2018, pp. 1–11.

[16] J. Hong, J. Kim, S. Han, and E.-Y. Chung, ‘‘A locality-aware compression scheme for highly reliable embedded systems,’’ *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 3, pp. 453–465, Mar. 2019.

[17] H. Jin, D. Jeong, T. Park, J. H. Ko, and J. Kim, ‘‘Multi-prediction compression: An efficient and scalable memory compression framework for GP-GPU,’’ in *Proc. IEEE Comput. Archit. Lett.*, Sep. 2022, pp. 1–29.

[18] J. Kim, M. Sullivan, and M. Erez, ‘‘Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory,’’ in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 101–112.

[19] J. Kim, M. Sullivan, E. Choukse, and M. Erez, ‘‘Bit-plane compression: Transforming data for better compression in many-core architectures,’’ in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 329–340.

[20] J. Kim, M. Sullivan, S.-L. Gong, and M. Erez, ‘‘Frugal ECC: Efficient and versatile memory error protection through find-grained compression,’’ in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2015, pp. 1–17.

[21] S. Kim, W. Kwak, C. Kim, D. Baek, and J. Huh, "Charge-aware DRAM refresh reduction with value transformation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 663–676.
[22] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 361–372.
[23] Y. S. Lee, G. Koo, Y.-H. Gong, and S. W. Chung, "Stealth ECC: A data-width aware adaptive ECC scheme for DRAM error resilience," in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, Mar. 2022, pp. 382–387.
[24] Y. Li and M. Gao, "Baryon: Efficient hybrid memory management with compression and sub-blocking," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 137–151.
[25] D. Lin, F. Yi, G. Yongliang, and M. Guizani, "Design of protograph LDPC-coded MIMO-VLC systems with generalized spatial modulation," *China Commun.*, vol. 21, no. 3, pp. 118–136, Mar. 2024.
[26] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, and G. Markosyan, "Synopsys' open educational design kit: Capabilities, deployment and future," in *Proc. IEEE Int. Conf. Microelectronic Syst. Educ.*, Jul. 2009, pp. 1–30.
[27] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Fault sim: A fast, configurable memory-reliability simulator for conventional and 3D-stacked systems," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 1–24, Jan. 2016.
[28] A. Navarro-Torres, J. Alastruey-Benedé, P. Ibáñez-Marín, and V. Viñals-Yufera, "Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel xeon skylake-SP," *PLoS One*, vol. 14, no. 8, Aug. 2019, Art. no. e0220135.
[29] A. Patil, V. Nagarajan, R. Balasubramonian, and N. Oswald, "Dvé: Improving DRAM reliability and performance on-demand via coherent replication," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, Jun. 2021, pp. 526–539.
[30] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "COP: To compress and protect main memory," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 682–693.
[31] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2012, pp. 377–388.
[32] E. Sha, A. Liu, K. Ibrahim, M. Mahmoud, C. Giannoula, A. Abdelhadi, and A. Moshovos, "Marple: Scalable spike sorting for untethered brain-machine interfacing," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2024, pp. 1–36.
[33] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "FinCACTI: Architectural analysis and modeling of caches with deeply-scaled FinFET devices," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 290–295.
[34] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in *Proc. 40th Asilomar Conf. Signals, Syst. Comput.*, 2006, pp. 1–22.
[35] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, Jun. 2012, pp. 285–296.
[36] Intel. (2021). *Intel Xeon Gold 6338 Processor*. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/212285/intel-xeon-gold-6338-processor-48m-cache-2-00-ghz/specifications.html

JI HUN KWON received the B.S. degree in electrical engineering from Hanyang University ERICA, in 2020. His research interests include error correction code and thermal management.

HYEONG KON BAE received the B.S. degree in electronic and information engineering from Sejong University, in 2021, and the M.S. degree from the Department of Computer Science, Korea University, in 2023. His research interests include error correction codes and memory systems.

YOUNG SEO LEE (Member, IEEE) received the B.S. and Ph.D. degrees from the Department of Computer Science, Korea University, in 2018 and 2022, respectively. He is currently an Assistant Professor with the School of Electronic Engineering, Soongsil University. Prior to joining with Soongsil University, he was an Engineer at Samsung Electronics, from 2022 to 2023. His research interests include 3D stacked architecture processing-in-memory, high-bandwidth memory, and reliable memory systems.

YOUNG-HO GONG (Member, IEEE) received the B.S. and Ph.D. degrees from the Department of Computer Science, Korea University, in 2012 and 2018, respectively. He is currently an Assistant Professor with the School of Software, Soongsil University. Prior to joining with Soongsil University, he was a Staff Engineer at Samsung Electronics, from 2018 to 2020, and an Assistant Professor at Kwangwoon University, from 2020 to 2023. His research interests include low-power memory design, 3D stacked architecture, processing-in-memory, and AI accelerator design.

SUNG WOO CHUNG (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, in 1996, 1998, and 2003, respectively. He is currently a Professor with the Department of Computer Science, Korea University. Prior to joining with Korea University, he was a Senior Engineer at Samsung Electronics, from 2003 to 2005. His research interests include system-level thermal/power management, system-level exploration of 3D stacking, and processing in memory. He was an Associate Editor of IEEE Transactions on Computers, from 2010 to 2015. He was the Technical Program Co-Chair of the IEEE International Conference on Computer Design, in 2015. He also serves (and served) on the technical program committees in many conferences, including DAC, from 2015 to 2018, ISLPED from 2016 to 2022, IPDPS from 2017 to 2018, and ICCAD in 2024.