Compression with Multi-ECC: Enhanced Error Resiliency for Magnetic Memories

Irina Alam*
irina1@ucla.edu
University of California, Los Angeles
Los Angeles, California

Saptadeep Pal*
saptadeep@ucla.edu
University of California, Los Angeles
Los Angeles, California

Puneet Gupta
puneetg@ucla.edu
University of California, Los Angeles
Los Angeles, California

ABSTRACT

Emerging non-volatile magnetic memories such as the spin-torque-transfer random access memories (STT-RAMs) provide superior density and energy benefits compared to conventional DRAM or Flash based memories. However, these technologies often suffer from reliability issues and thus strong conventional reliability schemes are required. These schemes have large overhead for storage which, in turn, can potentially eclipse the density and energy benefits these technologies promise. Moreover, the read and write operations in STT-RAMs show asymmetric behaviour i.e., bit-flip probability of 1⟶0 is significantly higher than 0⟶1. However, conventional Error Correcting Codes (ECCs) treat both 0 and 1 flips similarly and thus result in unbalanced reliability of these two types of errors. In this work, we propose a new ECC protection scheme for STT-RAM based main memories, compression with multi-ECC (CME). First we try to compress every cache line to reduce its size. Based on the amount of compression possible, we use the saved additional bits to increase the protection from the baseline Single Error Correcting, Double Error Detecting (SECDED) code to stronger ECC schemes, if possible. Compression itself reduces the hamming weight of the cache lines, thus reducing the probability of 1⟶0 bit-flips. Opportunistically using stronger ECC schemes further helps tolerate multiple bit-flips in a cache line. Our results show that for STT-RAM based main memories, CME can reduce the block failure probability by up to 240x (average 7x) over using a (72,64) SECDED scheme for each cache line word. The latency and area overheads of CME is minimal with average performance degradation of less than 1.4%.

CCS CONCEPTS

• **Computer systems organization → Reliability**; **Processors and memory architectures**; • **Hardware → Spintronics and magnetic technologies**.

KEYWORDS

STT-RAM, Cache line Compression, Multi-ECC

*Both authors contributed equally to this research.

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357533

ACM Reference Format:
Irina Alam, Saptadeep Pal, and Puneet Gupta. 2019. Compression with Multi-ECC: Enhanced Error Resiliency for Magnetic Memories. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3357526.3357533

1 INTRODUCTION

The criticality of memories in the design and performance of today's computer systems is becoming increasingly prominent. Main memories serve a pivotal role, sitting in between the processor cores and the slow storage devices. With aggressive technology scaling, a large number of processor cores are being integrated in today's systems. As a result, there is an ever increasing demand for main memory capacity in order to be able to exploit the processing power of these multicore and manycore systems and maintain the performance growth. However, DRAM scaling is unfortunately slowing down. Though DRAM is still the main memory workhorse, several application contexts need different properties from the main memory (higher density, non-volatility, higher performance, etc). Hence, it is becoming increasingly important to consider alternative technologies that can potentially avoid the problems faced by DRAM and enable new opportunities.

Several emerging non-volatile memory (NVM) technologies are now being considered as potential replacements for or enhancements to DRAM. Most of these new non-volatile technologies (Phase Change Memory[PCM], STT-RAM, Resistive RAM[ReRAM], etc.) promise better scaling, higher density, and reduced cost-per-bit [27]. However, they come with their own set of challenges. The biggest problem that these emerging technologies face is the high stochastic bit error rate. In fact, the reliability challenges of NVMs can offset the density and energy advantages that they offer. Increase in demand for memory capacity requires aggressive scaling of area-per-bit of storage. At higher density, these non-volatile emerging memory technologies tend to be more susceptible to stochastic bit errors [37]. Due to the random nature of the bit errors, these memory technologies require stronger in-field error-correcting code (ECC) [12].

The stochastic nature of failures in NVMs is similar to the radiation induced soft errors in DRAM and SRAM and occur without any warning. In order to ensure the integrity of the data, an error detection mechanism, followed by correction of the error(s) needs to be incorporated in a system. In conventional systems, ECC schemes are deployed to recover from memory errors. These schemes require adding redundancy bits alongside the original data (or message). For DRAM based memory, the most commonly used ECC schemes to recover from bit error or faulty chip error are the

SECDED (Single-Error Correcting, Double-Error Detecting) [14] and Chipkill-Correct [18] schemes.

This stochastic bit error rate in NVMs, however, is much higher than the single-bit soft error rate in DRAM. For example, in PCM, a two-bit cell may have 10^6-times higher error rate than DRAM and would, therefore, require a much stronger ECC scheme [12, 35]. Also, for most of these emerging NVM technologies, some states show higher error rates than the rest. As a result, the conventional ECC schemes used in DRAM-based memory need to be extended for providing multi-bit asymmetric protection to maintain acceptable limits of yield and performance of systems with these emerging memory subsystems. However, the cost and complexity of stronger error detection and correction circuitry increases exponentially, requiring much larger number of redundancy bits. This adds overhead not just in terms of storage but also power and performance.

Out of the various magnetic NVMs that have been proposed, Spin-Transfer Torque Random Access Memory (STT-RAM) is one of the most promising non-volatile technologies and has been studied extensively as a scalable non-volatile alternative to DRAM [9, 11, 15]. While STT-RAM might not have huge density benefits over DRAM like other technologies like PCRAM [4], its read performance is comparable to that of DRAM. The write energy and latency are roughly 5-10X and 1.25-2X respectively worse than that of DRAM [24, 25, 39] but much better than most other non-volatile technologies. Also, it has zero leakage power and much better program/erase endurance than the other competing NVM alternatives. In [24], the authors show that with certain optimizations, a STT-RAM main memory can achieve performance comparable to DRAM, while reducing the main memory energy by 60%, thus, making a strong case for STT-RAMs as a potential main memory alternative. STT-RAMs are also being considered as SRAM substitute for on-chip caches and has already been introduced in several commercial products [29, 36]. Though STT-RAMs are not susceptible to radiation induced soft errors, they suffer from a very high Bit Error Rate (BER) [40, 50]. As the NVM technology scales below 45nm, read disturbance error, retention error due to thermal instability, and write error rates are growing, leading to unacceptably high bit error rates (BER). Several circuit level and bit-cell design solutions have been proposed to lower the error rates [42]. Also, a few recent efforts have been made to provide stronger error resiliency [12, 23, 47]. Most of these solutions, however, result in very high energy and area overhead.

In this paper, we propose CME (Compression with Multi-ECC), a novel scheme to provide strong error correction in Magnetic RAM (MRAMs) based main memory subsystems. Though the proposed techniques would be useful for other types of DDR-based memories, we only consider the characteristics (and error-rates) of STT-RAM in our evaluations. This paper makes the following contributions:

- We use compression to reduce the size of each cache line so that the saved bits can be used to opportunistically add stronger protection without incurring the storage overhead of the redundancy bits of the stronger ECC codes. The code is chosen such that the final length of every cache line after compression and ECC remains constant in order to make the proposed CME scheme DDR compatible.
- Given the asymmetric nature of errors in STT-RAMs (explained in detail in Section 2), compression not only helps to reduce the length of the cache line, it also reduces the number of '1's (hamming weight) in each cache line.
- Since the final compressed cache line length can be anywhere between 13-bits (best case) and 512-bits (no compression), there is a wide variety of code choices available. We propose a dynamic programming solution to choose optimal mix of ECC codes to use, given the weight distribution of cache line words and the final cache line length distribution after compression. We show that optimized CME can achieve upto 240x (avg. 7x) reduction in block failure probability.
- We present two minimal memory architecture changes required to accommodate the tag chip that holds the tag required for our CME scheme per cache line and show that the performance overhead of CME is less than 1.4 % on average.

2 BACKGROUND

This section provides a brief background on two important concepts: the reliability concerns of STT-RAM based memories, and cache line compression techniques. STT-RAM memories suffer from high read/write/thermal error rates. A lot of these errors are asymmetric in nature, i.e., the probability of an error happening in a particular state is higher than the rest of the states. Cache line compression, on the other hand, has been used extensively in the past mainly to satisfy the rising demand for memory capacity and bandwidth. But in this work compression is used opportunistically for providing stronger error detection and correction to the cache lines.

2.1 STT-RAM Basics

In an STT-RAM cell, data is stored in a magnetic tunneling junction (MTJ). As current is passed through a mono-domain ferromagnet, the angular momentum of the electrons flips the direction of magnetization in the ferromagnet. The basic structure of a STT-RAM cell is given in Figure 1. MTJ consists of a tunneling oxide (MgO) separating two ferromagnetic layers. One layer (reference layer) has fixed magnetization and the other is a free layer whose direction of magnetization flips depending on the direction of current of sufficient density. The relative alignment of the two layers results either in a high resistance path (when opposite and usually represents '1') or a low resistance path (when parallel and usually represents '0').

Errors in STT-RAM can be broadly classified under three categories: read disturb errors, write errors and retention errors due to thermal instability.

2.1.1 Read Errors. The read operation in STT-RAMs is unidirectional. In STT-RAM, feature size scaling has led to a reduction in write current; however, read current has not reduced as much since the correct data may not be sensed when using low-current value. As technology scales below 45nm, read current doesn't reduce significantly beyond 20μA while the write current reduces to around 30μA [40]. Thus, read current is getting closer to the write current such that the read operation now has the potential to alter the stored value. Such an error is called read disturbance error. The data that is read is correct but the stored value becomes erroneous and subsequent reads from this location may contain multiple bit-flips. Since the read current is unidirectional, the unintentional bit

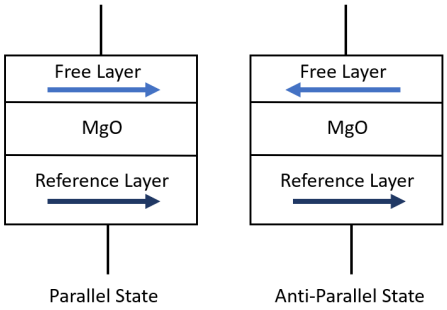

Figure 1: Schematic of STT-RAM showing the anti-parallel and parallel states

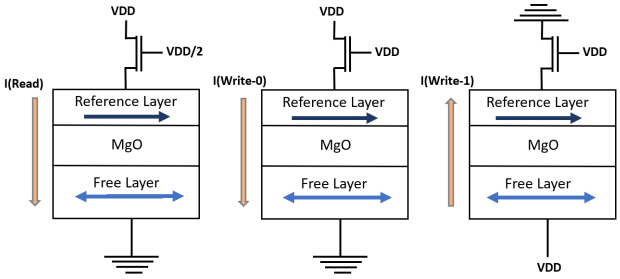

Figure 2: Read and write mechanisms for STT-RAM is shown here

flip during read is asymmetric and happens only in one direction ($1 \rightarrow 0$ when reading a '1'). Thus, reducing the number of 1's (or hamming weight) in a cache line will considerably help to reduce the read disturbance errors (RDEs).

2.1.2 Write Errors. In STT-RAM, a write failure happens if the switching current is removed before the MTJ switching completes. The time required for flipping the cell content varies due to the stochastic switching characteristics of the MTJ. However, this failure is also asymmetric [50]. When writing to STT-RAM cells, the MTJ switching from low-resistance state to high-resistance state ($0 \rightarrow 1$) is considered as "unfavorable" switching direction compared to the MTJ switching in the opposite direction: $0 \rightarrow 1$ flipping requires larger switching current than $1 \rightarrow 0$ flipping due to lower spin-transfer efficiency. Also, the variation of MTJ switching time at $0 \rightarrow 1$ flipping is more prominent. Hence, the chances of write error happening are much higher during a $0 \rightarrow 1$ transition than a $1 \rightarrow 0$ transition. As mentioned in [47], the bit error rate of $0 \rightarrow 1$ flipping is $P_{ER,0\rightarrow 1} \sim= 5 \times 10^{-3}$ while that of $1 \rightarrow 0$ flipping is $P_{ER,1\rightarrow 0} \sim= 10^{-7}$. They have also analyzed and concluded that the reliability of a word in a cache line decreases exponentially with increase in the hamming weight of cache lines. Thus, just like RDEs, Write Error Rate (WER) can also be reduced by reducing the hamming weight of a cache line.

2.1.3 Retention Errors. In STT-RAM, the third major source of errors is retention error where the data stored in the STT-RAM cell flips after a certain period of time. This false switching of data during the standby state is due to the inherent thermal instability of STT-RAMs. Increasing the thermal stability not only reduces retention errors but can also help to reduce read disturb errors [49]. But the critical current or the write current is proportional to the thermal stability of the cell. Higher thermal stability requires a higher write current and/or a longer write pulse. Thus there is a fundamental trade-off between write-ability (write time and/or power) and retention time [17]. Also, thermal stability of STT-RAM cells can be increased by using larger cell sizes, thus, increasing robustness at the cost of area [3].

2.2 Previous Work On STT-RAM Reliability

Errors due to read disturbance can be reduced using restore operation, which writes back the data every time there is a read operation [40]. Another work [32] suggests using a pulsed read technique to reduce read disturb errors in STT-RAM cells. However, all these techniques have significant overheads in terms of latency, energy and complexity. One recent work [28] suggests the use of data compression to enable duplication of bits in the memory. If cache lines are duplicated, then a restore operation would be needed only after all the copies have been read. This can potentially decrease the number of restore operations required after every read to deal with read error disturbances. However, if the STT-RAM based memory system uses DDR protocol, the entire cache line (both original and duplicated copies) would get read into the row buffer from the memory array for every read operation. Thus, duplication would technically not reduce the number of restore operations. There are also some bitcell architectures proposed [19, 51] to alleviate the problem of read disturb. However, they also incur significant overheads and only help in dealing with a single type of error. The authors in [6] propose a circuit level technique to detect read disturb errors but detection alone with no correction or reduction in error rates doesn't help to reduce performance degradation that happens due to system crashes when uncorrectable errors occur.

To deal with write errors, [47] suggests reducing the Hamming weight of each cache line. If the number of 1's is reduced in each line, it would considerably reduce the probability of having write errors since a $0 \rightarrow 1$ flip requires longer time and larger current and is thus, more prone to write errors. To reduce the Hamming weight of the cache line, [47] suggests using static/dynamic XOR between words of each cache line exploiting the value locality of stored data. Also, few recent works [12, 44] suggest improvements at the circuit level to improve BER of magnetic memories. Intel, in its recent STT-RAM work [46], proposes using a costly write-verify-write scheme to reduce write errors and a two stage current sensing technique during read to mitigate read disturb error. However, they still have significantly high write bit error rate (can be as high as 10^{-6}).

In [3], the authors proposed using stronger ECC in order to reliably decrease the size of STT-RAM cells and have shown that higher density can be achieved if stronger ECC protection is used. Another work [42] proposed adaptive write-schemes using in-memory variation sensors to reduce write latency reliably for better application

performance. To deal with read margin errors under thermal variation, [48] designed a body-biasing feedback circuit to improve read sensing margin for STT-RAMs. Most of the proposed techniques either target mitigation from one type of error (write or read error) or have very large overheads in terms of circuit complexity, area or power.

2.3 Previous Work On Cache Compression

Cache line compression techniques are being widely proposed to satisfy the rising demand for memory storage capacity and memory bandwidth [1, 33, 41]. These techniques exploit spatial and value locality of the data in typical applications. In most cases, there are only a few primitive data types supported by hardware (e.g., integers, floating-point, and addresses), which typically come in multiple widths (e.g., byte, halfword, word, or quadword) and are often laid out in regular fashion (e.g., arrays and structs). The data used in most applications, on the other hand, are low magnitude and are often represented inefficiently, for e.g., 4-byte integer type used to represent values that usually need only 1-byte.

One commonly used compression scheme is Base-Delta-Immediate (BΔI), as proposed in [31]. This scheme exploits the low dynamic range of values present in many cache lines to compress them to smaller sizes. They split up the cache line into multiple equal sized chunks. They take the first chunk as the base and represent all the subsequent chunks as delta with respect to the base. The delta value is smaller than the original value due to the existing value locality in the cache line and hence, can be represented using lesser number of bits. While the authors manage to reduce the compression and decompression latency as compared to the popularly used cache compression techniques such as Frequent Pattern Compression [1], the increase in compression ratio is not significant.

Another recent work on cache compression [20] claims to have better compression ratio than BΔI. This work exploits locality in two layers: within values or words of the cache line data and within bits in the same bit-plane. A bit-plane is a set of bits corresponding to the same bit position within each cache line word in a data array. As a result they manage to achieve higher compression ration than most of the previously proposed cache compression techniques. However, this scheme has been proposed for 128-byte cache lines and needs to be modified for systems with 64-byte cache lines. Both BΔI and BPC schemes have been compared later in Section 6.1 and the pros and cons of each compression scheme in the context of stronger error detection and correction have been discussed.

Most of the past works utilize cache compression to effectively increase the size of the cache. However, our goal is to utilize compression to reduce the hamming weight of the cache lines and also to utilize the additional space to opportunistically add in stronger error correction codes (ECC). Compression with ECC has been proposed previously in the context of DRAM based memory system in FrugalECC [21], COP [30] and Free ECC [10]. In [21] and [10], they used the same protection for every cache line that could be compressed beyond a certain threshold. If uncompressed, the overflow data required additional storage and accesses. In [30], they added error correction only when compression was possible, leaving some cache lines unprotected because of lack of compressibility. Also they use the same protection for every compressed cache line irrespective of how much a particular cache line could be compressed. Another problem with these schemes is that their compression ratio goals are very modest since they focus on metadata. In our case we use different ECC schemes for different cache lines depending on the final compressed size of that particular line. Thus, after encoding with ECC, every cache line is of uniform size and therefore, has no overflow requiring extra storage and accesses. Also we can opportunistically provide much stronger protection to sufficiently compressed cache lines.

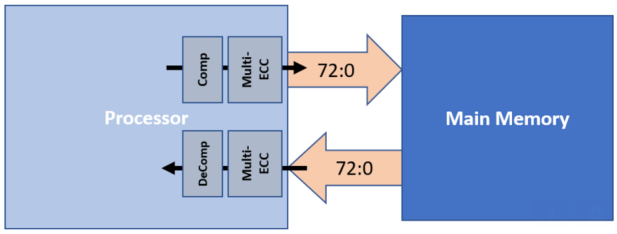

Figure 3: Processor Memory system architecture with CME

3 OUR SCHEME - COMPRESSION WITH MULTI-ECC (CME)

In this section, we will discuss the details of Compression with Multi-ECC (CME) scheme. Cache line compression is used for two reasons. Firstly, it helps in reducing the hamming weight of each cache line. Secondly, it enables either data duplication (when the compressed cache line is less than half its uncompressed size) or allows to use the available bits to provide stronger error protection. The selection of the stronger ECC scheme depends on the size of the cache line post compression such that the final size of each cache line with the redundant bits remains uniform.

3.1 Overall Architecture

As shown in Figure 3, every time a cache line is to be stored in the memory, it is a two-step approach. It first goes through the compression engine and then through the Multi-ECC encoder. In case of a load operation, it first goes through the ECC Decoder and then the de-compression engine. Even though the size of an ECC word is 72-bits, the 73rd bit shown in Figure 3 is to signify a tag bit that is sent across the DDR bus in every cycle.

We used a slightly modified version of Bit-Plane Compression (BPC) scheme proposed in [20] which is explained in detail in the following subsection.

3.2 Cache Line Compression using modified BPC and an optional Hamming Weight Aware Inversion Coding

Bit Plane Compression (BPC) as described in [20] is a two-step process on a 128B cache line, the first step is where the data is transformed to increase compressibility and the second is to encode the transformed data. We slightly modified the compression scheme for a 64B cache line. We used it for a 64-bit architecture where each

cache line word is 64 bits. However, the compression is done on 32-bit words. So we split each 64-bit word into two 32-bit sub-words, the higher order bits (bits 63-32) constitute one sub-word and the lower order bits (bits 31-0) form the other sub-word. The first step is cache line manipulation and transformation (Delta-BitPlane-XOR [DBX]) to improve compressibility of data and thus reduce the compression hardware complexity.

The next step after data transformation is the compression of the transformed data. BPC combines run-length encoding with a type of frequent pattern encoding to compress the transformed data. As mentioned before, the work in [20] used word-size of 64 bits in a 128-byte cache line, while for our evaluations we use 32-bit words and 64-byte cache line. Hence, our symbol encoding is slightly different from theirs and is shown in table 1. This encoding scheme not only helps to reduce the cache line length but also helps to reduce the hamming weight considerably (as seen in Figure 11). For instance, in this encoding scheme, a running length of 1's gets encoded to {5'b00000}. The base (first two original) symbols is compressed separately by original symbol encoder as {3'b000}, {3'b001, 4-bit data}, {3'b010, 8-bit data}, or {3'b011, 16-bit data} if its value is 0 or fits into 4/8/16-bit signed integer, respectively. Otherwise, the base symbols are encoded as {1'b1, 32-bit data}.

Since both read and write errors in magnetic memories is asymmetric, reduction of hamming weight is important for resiliency. After each BPC word is encoded, an optional inversion coding [38] can be added to further reduce the hamming weight of the cache line. We check the weight of each BPC encoded word. If it is greater than half the size of the word, we invert the word. In order to facilitate inversion encoding, we add one additional bit in front of each BPC word ('1' if the word is inverted and '0' otherwise). However, note that the additional bit per BPC word required for inversion increases the length of the final compressed line. In Section 5.2 we show that, in many cases, this increase in cache line size due to inversion tag bits leads to weaker ECC scheme and finally, in spite of reducing the hamming weight, inversion ultimately adversely affects the overall block failure probability. One way to reap the benefits of hamming weight reduction using inversion coding without significantly increasing the size of the cache line would be to apply inversion on groups of multiple BPC words instead of a per word basis.

After doing BPC and inversion coding we check the final encoded size of the entire cache line. If the length exceeds 512-bits, the raw cache line is taken and encoded with a (72,64) SECDED code before writing to the main memory. If the compression successfully reduces the size of the cache line, we opportunistically encode the cache line with a stronger ECC code before writing to the memory.

Table 1: Frequent Patterns for BPC and DBP/DBX symbol encoding

DBP/DBX Pattern	Length	Code (binary)
0 (run-length 2~33)	7-bit	{2'b01, (RunLength-2)[4:0]}
0 (run-length 1)	3-bit	{3'b001}
All 1's	5-bit	{5'b00000}
DBX!=0 and DBP=0	5-bit	{5'b00001}
Consecutive two 1's	9-bit	{5'b00010, StartingOnePosition[3:0]}
Single 1's	9-bit	{5'b00011, OnePosition[3:0]}
Uncompressed	16-bit	{1'b1, UncompressedData[14:0]}

3.3 Multi-ECC on Compressed Cache Line

Compression helps to reduce the size of the cache line in most cases. Once the reduction is done, the final size of the cache line determines the ECC scheme to be used. Shorter the length of the compressed cache line, more the available redundant bits for ECC and therefore stronger will be the protection. We wanted to ensure that the STT-RAM based memory subsystem closely matches the standard ECC-DDR protocol. Hence, every fetch will be 72-bit wide.

3.3.1 Choice of Codes. The possibility of ECC schemes for a given redundancy is large. Since the final compressed cache line length can be anywhere between 13-bits (best case) and 512-bits (no compression), there is a wide variety of code choices available. Usually, each ECC word in DDR memory is comprised of 72 bits (original message + ECC bits) which is equal to the length of one fetch. However in CME, we also consider ECC code word sizes of 36 (i.e., two ECC words per fetch) and 144 (i.e., one ECC word per two fetches). We therefore restricted our code space to the options provided in Table 2.

Table 2: Choice of Error Correcting Codes for CME

ECC scheme	Length of ECC word (original message + ECC bits)
SECDED (Single Error Correcting, Double Error Detecting) [14]	36 (29 + 7) 72 (64 + 8) 144 (135 + 9)
DECTED (Double Error Correcting, Triple Error Detecting) [8]	36 (23 + 13) 72 (57 + 15) 144 (127 + 17)
3EC4ED (3 Error Correcting, 4 Error Detecting) [7]	36 (17 + 19) 72 (50 + 22)

The reason for using code word sizes of 36-bits and 144-bits alongside 72-bits is to increase the opportunity for stronger protection. As an example, let's assume a compressed message is 58 bits long, therefore splitting the message to two 29-bit messages can enable SECDED protection (29+7) on each 29-bit message and two code words can constitute a 72-bit fetch. If we had restricted our code space to 72-bit words, the 58 bits of the two messages will only get SECDED protection since DECTED protection will require the message to be of length 57 bits. It can be noted from Table 2 that, no code stronger than 3EC4ED (3 error correcting, 4 error detecting) has been used even for cases where stronger protection is possible (for eg. 4EC5ED can be used in cases where the compressed cache line size is less than 368 bits). This is because stronger ECC not only adds greater hardware complexity and overheads, the redundant bits added to each word in the cache line often increases the Hamming weight of the overall word considerably, thus increasing chances of read disturb/write errors. As seen in Figure 5, with stronger ECC, the block failure probability decreases rapidly till 3EC4ED, beyond which the benefit of stronger ECC saturates. 144-bit 3EC4ED was not evaluated because of the large size of the encoder/decoder alongside the large number of redundancy bits.

We also wanted to limit our tag overhead per cache line to 8. If the tag is protected by SECDED, then 4 bits of redundancy would be required. Thus the actual tag can be at most 4 bits. The first bit would be used to denote if the cache line is compressed or not. With 3 bits of tag left to denote the ECC scheme used on a compressed cache line, there can be eight distinct codes that can be used. We

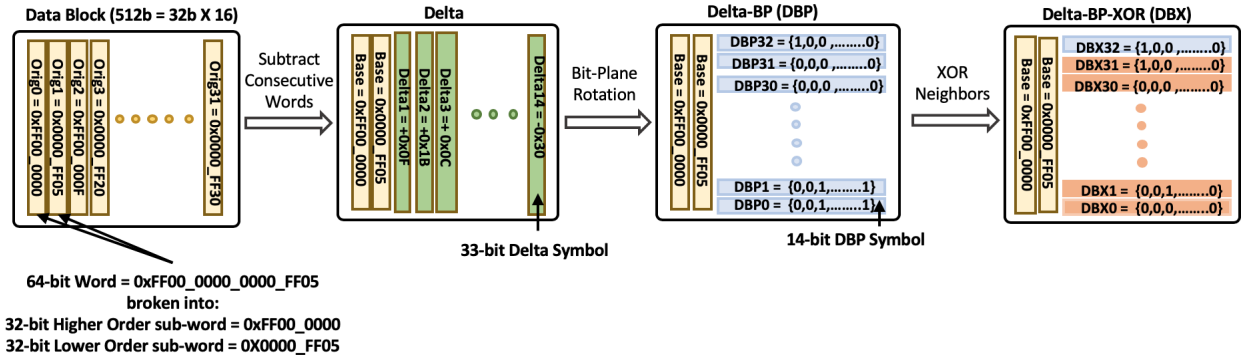

Figure 4: An overview of the modified Bit-Plane Transformation scheme

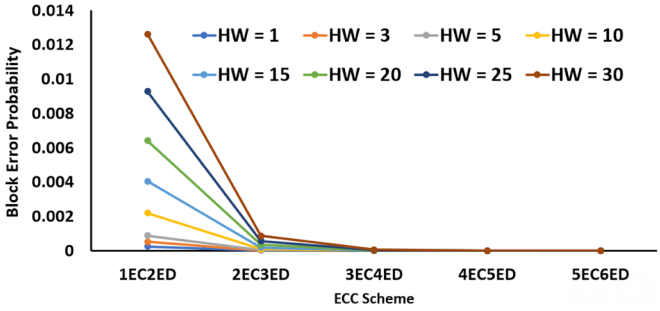

Figure 5: Block failure probability is shown for blocks with different Hamming weight (HW) and ECC schemes. The probability of 1→0 bit-flip is considered to be 10^{-5}

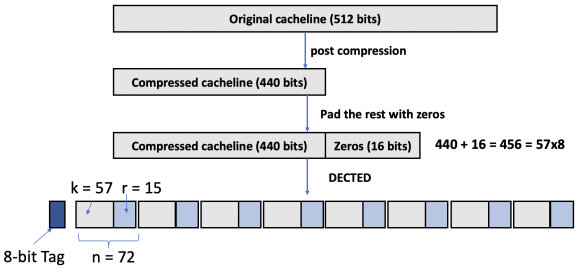

Figure 6: An example of CME scheme where the compressed cache line size is 440 bits

decided to use 2x and 4x duplication for any 512-bit cache line that gets compressed beyond 256-bits or 128-bits respectively. Therefore, cache lines with compressed length higher than 256 are left with six choices of codes.

Unlike [28] which uses duplication to avoid reading the entire line (which in standard DDR memories actually does not help with read disturb as explained earlier), we use it to minimize uncorrectable errors. As observed from our SPEC2006 benchmark traces, the average number of read operations between two write operations is less than two. Thus, when the data is duplicated once, a cache line word would fail only when both copies of the word have un-correctable errors (note that all used ECC codes detect more errors than they can correct). Therefore, in our scheme, we read the entire cache line (with all the copies) and only use the subsequent copies if there is an error in the first copy. Reduction in errors means scrubbing (refresh with ECC) operations will be needed less frequently, which saves both time and energy. If the compressed size is less than $1/4^{th}$ of the original size, we propose a 4x duplication for even stronger protection against errors.

Since all the cache lines within a certain range of compressed length get the same type of protection, majority of those cache lines would have to be padded with zeros at the end, before adding ECC, to increase the final message length to a certain value required by that particular code type. For example (shown in Figure 6), if all the cache lines whose final compressed length lies between 416 and 456 get (72,57) DECTED protection, all cache lines within that range whose length is not exactly equal to 456 would need to be padded with zeros to increase the final length to 456 before splitting up the cache line into eight 57-bit words and adding DECTED protection on each one of them.

Based on this we realized that for any range, most of the cache lines would have the last few words being all 0s. This can be seen in Figure 7 where we show the hamming weight of each 32-bit word of compressed cache lines. Based on the compressed cache line size, we bin the cache lines with post-compression size in between 256 to 512 bits in to four buckets. Each plot in the figure corresponds to one bucket and shows the distribution of average hamming weight of the 32-bit words across six benchmarks from the SPEC2006 suite. Each of these 6 benchmarks had widely different average compression ratio (original-size/compressed-size) and hamming weight. Hence it can be argued that these six benchmarks are good representations of the entire suite.

It can be seen that in most cases, the last few words have much less average hamming weight as compared to the rest. This is because majority of those words have all 0s. Thus, we decided to add split codes to our code space. In split codes, the same cache line would have two different types of protection. The first few words would have stronger protection than the last few. In order to limit the code space, we decided to evaluate combinations of at most two codes from Table 2 and limit the use of weaker protection up till

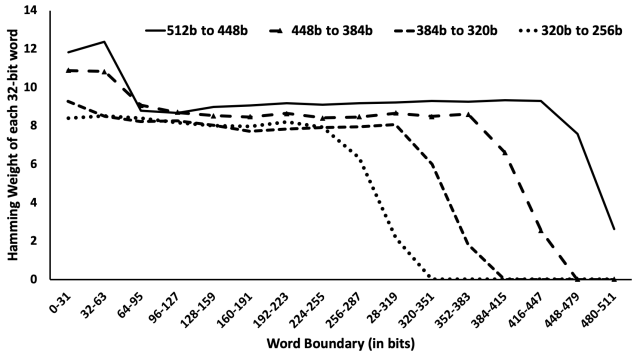

Figure 7: Average hamming weight of each 32-bit word of all cache lines within each bucket. Uniform bucket size of 64 bits were used for all cache lines whose final size lies between 512 bits and 256 bits.

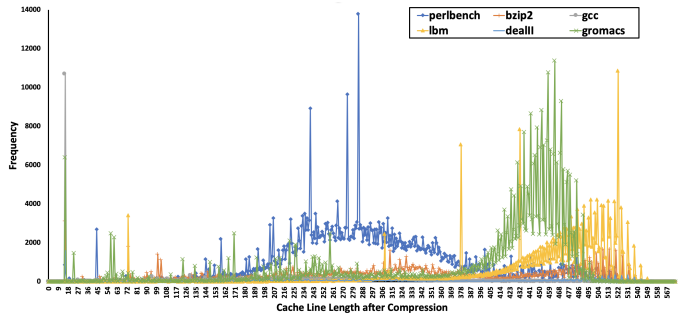

Figure 8: Distribution of cache line length after compression of six benchmarks from the SPEC2006 suite

the fourth word from the end. To further limit the code space, after listing down all the choices, we chose the strongest code within a redundancy range of 4 bits. This means that if by adding 4 extra bits we would get a strictly stronger code, we would remove the weaker one from our final evaluation. Overall, after all these manipulations, we were left with 28 codes (from the possible 64 code pairs) from which we would have to select the best six codes. Next, we discuss how we select these best six codes which constitutes CME.

3.3.2 Dynamic Programming to choose the final set of codes. To select the best six codes, we took into account the final distribution of cache line length after compression and the average hamming weight of each word for all cache lines within a certain range of compressed length across the SPEC-2006 benchmark suite. The same six benchmarks mentioned previously were also used for this. The cache line length distribution of the six benchmarks can be seen in Figure 8.

Based on these distributions, we used dynamic programming to choose the six optimal codes.

For each code, the maximum compressed length of a cache line for being protected by that code is fixed. For example, for each of the 8 words in a cache line to have DECTED protection, the compressed cache line can be at most 456 bits. This maximum length is fixed for each type of code.

For every code choice, we first find the weight distribution of all cache lines whose compressed length is within the range (`max_length`, 256), where `max_length` is the maximum allowable length for that particular code. We use this weight distribution to calculate the block failure probability if this code was selected for these cache lines. The failure probability is then weighted by the fraction of cache lines that fall within this range to evaluate the effectiveness of adding the code.

For the dynamic programming, we start with the code (say codeA) which has the smallest `max_length` and calculate the block failure probability over the cache lines of compressed size (`max_lengthA`, 256). Next, we add the second code (say codeB) with the next smallest (`max_length`). We then evaluate two cases, (1) When codeA and codeB are both used together to protect cache lines between (`max_lengthA`, 256) and (`max_lengthB`, `max_lengthA`) respectively; calculating this joint probability can leverage the block probability for (`max_lengthA`, 256) that was calculated in the last step and (2) When only codeB is used.

In subsequent iterations, we add new code types and calculate the combined block failure probability by leveraging the already calculated block failure probabilities for smaller `max_length` codes. From the seventh iteration onwards, we only calculate the probabilities of code combinations which has six codes in total. Since most of the code combinations except the newly added code has been evaluated in previous iterations, the dynamic programming approach helps us minimize the time to calculate the block failure for a set of six codes. After all the six code combinations are iterated through, we choose the one with the smallest aggregate block failure probability. The set of codes that are finally chosen are given in the Table 3.

Table 3: ECC scheme to be used depending on the compressed cache line size

Length of compressed cache line (in bits)	ECC scheme to be used
>512	No compression (Use Raw Cache line + (72,64)SECDED)
≤512 and >508	(72,64)SECDED on each 64-bit cache line word
≤508 and >495	(144,127)DECTED on each 127-bit cache line word
≤495 and >482	(72,57)DECTED on the first 2 57-bit cache line words and (144,127)DECTED on the last 3 127-bit cache line words
≤482 and >469	(72,57)DECTED on the first 4 57-bit cache line words and (144,127)DECTED on the last 2 127-bit cache line words
≤469 and >427	(72,57)DECTED on the first 6 57-bit cache line words and (144,127)DECTED on the last 127-bit cache line word
≤427 and >256	(72,50)3EC4ED on the first 6 50-bit cache line words and (144,127)DECTED on the last 127-bit cache line word
≤256 and >128	Duplicate the cache line (2 copies) and (72,64) SECDED on each word
≤128	Duplicate the cache line (4 copies) and (72,64) SECDED on each word

Note that this dynamic programming based code space search is done just once to find the best six codes which constitutes CME. A fixed CME scheme would be built in to the hardware and all the applications would use the same scheme.

3.4 Additional Tag Bits and Memory Organization

Every cache line now needs additional tag bits to denote if the cache line is compressed and what protection scheme is used.

Table 4: 8-bit Tag per Cache Line for CME

Tag Bits	When Compression is possible		When Raw Cache line is used
Bit-0	-	'1'	'0'
Bits1-3	BPC + (72,64)SECDED	'000'	'000'
	BPC + (144,127) DECTED	'001'	
	BPC + (72,57) DECTED on first two words and (144,127) DECTED on the rest	'010'	
	BPC + (72,57) DECTED on first four words and (144,127) DECTED on the rest	'011'	
	BPC + (72,57) DECTED on first six words and (144,127) DECTED on the rest	'100'	
	BPC + (72,50) 3EC4ED on first six words and (144,127) DECTED on the rest	'101'	
	BPC + duplication (2copies) + (72,64)SECDED	'110'	
	BPC + duplication (4copies) + (72,64)SECDED	'111'	
Bits4-7	(8,4)SECDED redundancy for the first 4 Tag bits	-	'0000'

As shown in Table 4, we use 8 additional bits of tag to each cache line to denote the transformation operation that was done for that particular cache line.

- *Bit0:* Denotes if the stored cache line has been compressed or not. If compressed then the first bit of the tag is '1'; else '0'.
- *Bits1-3:* When the cache line is compressed, these three additional bits denote the ECC/duplication scheme used for that cache line as given in Table 4, else the field is populated with '000'. Note that the tag bits for the duplication cases (last two) are the highest weighted since they are the least frequently occurring.
- *Bits4-7:* These 4-bits are ECC bits used to provide a (8,4) SECDED protection on the first 4-bits of tag to correct a single-bit error and detect any double-bit error.

3.4.1 DDR4 Primer. In today's system with DRAM based main memory system, a processor accesses the main memory through the memory controller. Memory controller buffers memory access requests from the processor, schedules the requests, converts them into DRAM commands complying with the specific DDR protocol and sends them over a DDR bus to the dual in-line memory module (DIMM). One or more DIMMs is supported on a memory bus. Each DIMM has a DIMM controller along with 9/18/36 x4/x8 DRAM memory chips. The DIMM controller acts as the interface to the DDR bus and manages the DIMM. In our example we use a DIMM with 9 x8 memory chips as the baseline to explain the modifications required to fit an additional STT-RAM tag chip along with 9 x8 STT-RAM chips on a STT-RAM DIMM connected to a DDR4 bus.

In a conventional DRAM based main memory system with x8 DRAM DIMM, for reading from or writing a cache line to the memory, each memory chip from the same rank in the DIMM sends 64-bits over 8 cycles to form a 512-bit (576-bits with ECC) cache line. First, the memory controller sends an ACTIVATE request to the DIMM along with the rank, bank and row addresses. Based on the addresses the DIMM controller activates the row in the bank of all the chips in that rank. The row is read from the array into the respective row buffers. Each row buffer now holds one row, i.e., an entire DRAM page. If it is DDR4 type memory one DRAM page size of an x8 memory chip is equal to 1KB. From this 1KB page per DRAM chip, only 64-bits need to be accessed sequentially in 8 bursts where each burst consists of 8-bits. The beginning of

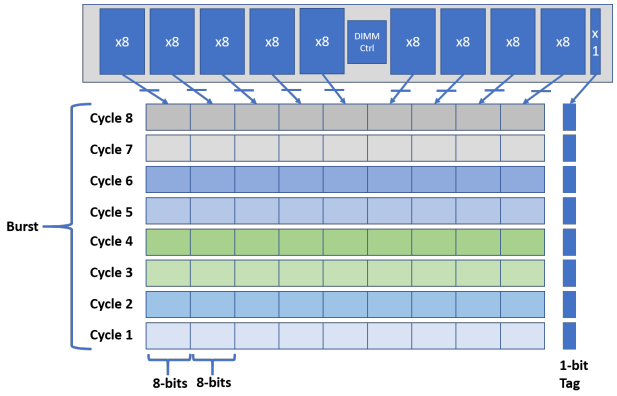

Figure 9: CME-Scheme 1 is shown where tag bits are stored in an x1 DRAM chip. One tag bit is read every cycle in burst. Different colors represent different 72-bit ECC words in a 512-bit cache line.

this 64-bit chunk in the 1KB page is determined by the column address sent to the DIMM by the memory controller next along with the READ/WRITE command. For a READ operation the last three bits of the column address determines the burst order, i.e., the order in which the cache line words will be read. This is to enable "priority word first" [26] for improving performance where the payload word gets read and decoded first and sent to the processor while the rest of the words are brought to the cache. The rest of the bits in the column address determine the beginning of the chunk being accessed.

Assuming the STT-RAM based main memory system uses DDR4 protocol, we explore two ways of storing the additional bits of tag per cache line.

3.4.2 Scheme 1. The first option is to store the tag bits separately in the memory. We require an extra tag chip to store the 8 bits of tag per cache line and an additional data signal in the DIMM. This extra tag chip will be a x1 memory chip with one data signal pin. The size of the memory page will be one-eighth that of the other memory chips in the DIMM (128B) and in each burst only one bit will be read instead of the conventional 8-bit burst. Thus, when the DIMM controller sends the READ/WRITE request to the tag chip, it will shift the column address bits by 3 (divide by 8). The data signal pin of this x1 tag chip will be connected to the extra data signal in the DIMM. The minor changes required to accommodate this extra tag chip are depicted in Figure 9.

Though implementing this only requires subtle changes to the DIMM and memory architecture, this scheme incurs latency overhead. In today's systems where the main memory has (72,64) SECDED protection, each 72-bit word read from the memory in a burst has to go through the ECC decoder to get the original 64-bit message and to check for any single-bit/double-bit errors. Since a 576-bit cache line (512-bits of message + 64 bits of ECC) is sent over 8 bursts, each of length 72-bits, and each burst is a separate ECC word, the ECC decoding can begin as soon as the first burst or ECC word arrives. However, in our case we have to wait for the tag bits before we can start with the ECC decoding and in every burst

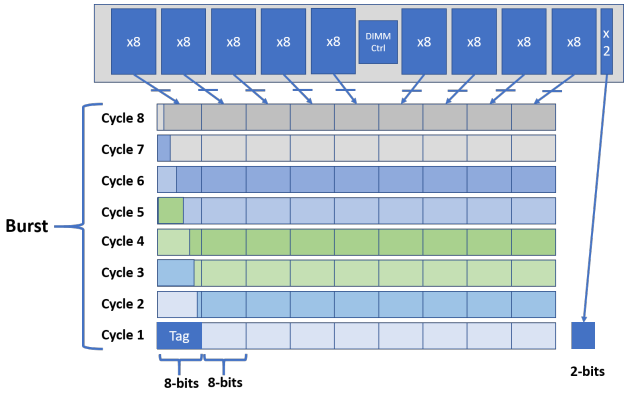

Figure 10: CME-Scheme 2 is shown where tag bits corresponding to ECC scheme used are stored in an x8 DRAM. The tag bit and it's parity representing if the cache line is compressed are stored in an x2 DRAM chip and are brought in the same burst. Different colors represent different 72-bit ECC words in a 512-bit cache line.

only one bit of tag is sent. This latency overhead will vary from cache line to cache line. For an uncompressed cache line, as soon as the first bit of tag arrives, the ECC decoding can start since all un-compressed cache lines are protected by (72,64)SECDED code. However, if the cache line is compressed, i.e., if the first tag bit is '1', the ECC decoding has to wait for at least 4 cycles. Since the tag is protected by a linear (8,4) SECDED code, the first 4 bits of tag are the original tag message and the last 4 bits are the redundancy bits. So, as long as there is no error in the tag bits, the first 4 bits of tag should be sufficient to tell us the ECC scheme used for that particular cache line. Thus, the ECC decoding can start after 4 cycles. If after 8 cycles it turns out that there was an error in the first 4 bits of tag then the ECC decoding has to be done again, causing a 8-cycle latency overhead, however probability of that happening is very small and will not impact the performance of a system.

3.4.3 Scheme 2. To avoid this latency overhead, we propose an alternative option of embedding the 8 bits of tag into the compressed cache line itself. This means that after encoding the compressed cache line with ECC, the final length has to be 568 bits so that adding 8 bits of tag to the cache line increases it to 576 bits (as shown in Figure 10). If the cache line cannot be compressed at all no tag bits will be needed as the standard (72,64) SECDED code will be used for all uncompressed cache lines. However, overall one bit of tag is still required separately to denote if the cache line has been compressed. The one extra bit of tag can either be stored in the memory controller for a small sized main memory system or a separate x2 memory chip in the DIMM can be used. If stored in the memory then an additional bit of parity is required to protect that one tag bit (1 bit is enough since errros are asymmetric). This tag bit can be fetched using a x2 memory chip in one burst and will have a page size of 4B. For the x2 memory chip, the least-significant 5-bits of column address select need to be ignored because of the reduced page size and no requirement of bursts. Reduction of the burst size will be minimal modifications of the circuitry that is present in today's DDR4 DIMM which supports variable burst sizes of 4 and 8. The DIMM will require 2 additional data signal pins and the memory controller needs to know that valid data will be sent over those 2 data signal pins only in the first burst. All these changes should be minimal and easy to implement given the current existing architectures. This scheme will have latency overhead of only 1 cycle. However, having this option will require our ECC schemes to change since now the final size of the ECC encoded message will be different. We re-ran our code selector to select the six optimal schemes and the selected codes are provided in Table 5. We call this CME Scheme-2 when we evaluate it in Section 5.2.

Table 5: ECC scheme to be used depending on the compressed cache line size when the tag is embedded in the cache line

Length of compressed cache line (in bits)	ECC scheme to be used
>504	No compression (Use Raw Cache line + (72,64)SECDED)
≤504 and >500	(71,63)SECDED on each 63-bit cache line word
≤500 and >487	(142,125)DECTED on each 125-bit cache line word
≤487 and >474	(71,56)DECTED on the first 2 56-bit cache line words and (142,125)DECTED on the last 3 125-bit cache line words
≤474 and >461	(71,56)DECTED on the first 4 56-bit cache line words and (142,125)DECTED on the last 2 125-bit cache line words
≤461 and >419	(71,56)DECTED on the first 6 56-bit cache line words and (142,125)DECTED on the last 125-bit cache line word
≤419 and >252	(71,49)3EC4ED on the first 6 49-bit cache line words and (142,125)DECTED on the last 125-bit cache line word
≤252 and >126	duplicate the cache line (2 copies) and (71,63) SECDED on each word
≤126	duplicate the cache line (4 copies) and (71,63) SECDED on each word

If the STT-RAM based memory is embedded and on-chip (as seen in some commercial STT-RAM based memory systems [13]), the tag placement will be much simpler and will not incur any additional latency overhead since all 8 bits can be fetched in one cycle.

4 EVALUATION METHODOLOGY

While compression and error correcting codes are used individually in caches and main memory, we focus on combining the two and opportunistically providing stronger protection for STT-RAM (or any other magnetic or high error rate memory) based main memory systems in this paper. We first compare the hamming weight reduction achieved after doing Bit-Plane Compression with and without hamming-weight-aware inversion against another scheme [47] proposed earlier to reduce hamming weight and an alternative compression scheme (BΔI [31]). We then evaluate the effectiveness of using stronger codes for STT-RAM based memories with error rates provided in Table 6. We use two design points, one from Intel [46] and the other from Samsung [22], for our analysis. For the Intel design point, the read/write/retention error rates are as provided. For the Samsung design point, the write error rate is provided along with the thermal stability factor (Δ). Based on the technology node, the read disturb rate is calculated from [15]. The retention error rate is calculated based on the Δ using Equation 1.

$$P_{ret} = 1 - e^{-\frac{t_r}{\tau_0} e^{-\Delta}} \quad (1)$$

where τ_0 is reversal attempt period and is on the order of a nanosecond [3], and t_r is the interval for which the evaluation is conducted.

For STT-RAM based memories, refresh is not the same as DRAM [3, 15]. In DRAM, refresh is used to prevent deterministic errors cause by charge leakage over time. But in STT-MRAM, errors are stochastic in nature. Therefore, if DRAM like refresh is performed in STT-RAM, where the cell contents are simply read and written back without any error correction, already flipped bits in the memory would be read and written back, as is, without any correction. This would not be effective in lowering the error rate. In STT-RAM, refresh needs to be accompanied by error correction and is similar to the scrubbing operations performed in today's systems. For all our analysis, unless otherwise mentioned, we consider a scrubbing interval of one second. We finally evaluate the hardware overhead of compression with multi-ecc and its impact on performance.

Table 6: Evaluation setup

Cache Line Size	512-bits (64-Byte)	
Design Point - I [46]	Write Bit Error Rate	1×10^{-6}
	Read Bit Error Rate	1×10^{-12}
	Retention Error Rate	Negligible (200C 10 years)
Design Point - II [22]	Write Bit Error Rate	1×10^{-6}
	Read Bit Error Rate	1×10^{-10}
	Retention Error Rate	Calculated based on thermal stability factor Δ = 40 using equation 1

To evaluate our protection scheme for STT-RAMs, we extracted memory traces of 18 benchmarks from the SPEC CPU2006 benchmark suite using Gem5 [5]. These applications are a mix of integer and floating point benchmarks. Next, each application was subjected to CME. The average block failure probability (average failure probability of each word in the cache line) was computed for each design point based on the final set of cache lines obtained after applying compression to the obtained memory traces.

The probability of a cache block/word of hamming weight W not failing under a certain write/read/retention bit error rate P_{ER} protected by a t-error correction ECC is given by the following equation:

$$P_{block} = \sum_{i=0}^{t} \binom{W}{i} (1 - P_{ER})^{W-i} (P_{ER})^{i} \quad (2)$$

For overall block error rate, we calculated the probability of failure using the knowledge obtained from the memory traces about the number of reads between two consecutive write/scrubbing instructions to a a particular memory address. For example, when a cache line protected using (72,57) DECTED code is read twice consecutively before a write operation to the same address, the probability of no fault is calculated by considering all the following cases: (a) When two or less faults occur during the same operation (either during write, any of the two reads or because of retention error). (b) One fault occurs during one operation and another fault occurs during another operation. When a cache line gets duplicated twice, we consider a block to be failing only when both copies have un-correctable errors. Based on our memory trace statistics, we saw that most of the cache lines are read once. However, there are still 2-5% of cache lines that are read more than twice (some even more than 10 times).

To evaluate the area and latency overheads of stronger ECC codes, we synthesized both (72,64) SECDED and (144,127) DECTED decoding engines using an industrial 45nm library. The (144,127)DECTED decoder is expected to have the largest overheads compared to the baseline (72,64) SECDED. To evaluate the performance impact of CME due to the overheads of compression and multi-ECC (discussed in detail in Section 5.4), we ran performance simulation using Gem5 [5]. Two micro-architectural configurations were evaluated as provided in Table 7. The first set of evaluations were done for a system with 8 in-order (InO) cores sharing only 2MB of unified L2 cache and no prefetch. So that there is minimal memory access latency hiding techniques. Also the shared L2 cache is not statically partitioned and the different processes compete for cache space. These are expected to exaggerate the effect of CME overheads on performance. For these experiments, each thread runs the same application in separate processes. The second set of evaluation was done for a single out-of-order (OoO) core having a unified 2MB L2 cache and with prefetching enabled. The performance evaluations on Gem5 were done for the benchmarks in the SPEC2006 suite, fast forwarding for 1 billion instructions and executing for 2 billion instructions. The latency overheads considered for the simulations are discussed in detail in Section 5.4.

Table 7: Core Micro-architectural Parameters

	Config-1	Config-2
Cores	8, InO (@ 2GHz)	1, OoO (@ 2GHz)
ISA	ALPHA	x86
L1 Cache per core	32KB I$ 32KB D$ 4-way	32KB I$ 32KB D$ 4-way
L2 Cache	2MB (shared, unified) 8-way	2MB (unified) 8-way
Cache Line Size	64B	64B
Memory Configuration	32GB of single-channel x4 DDR4-2400	32GB of single-channel x4 DDR4-2400
Nominal Voltage	1V	1V

5 RESULTS

In this section we demonstrate that CME provides considerable benefit in terms of block error reduction as compared to a normal (72,64) SECDED code. The evaluations also show the minimal impact of the hardware overheads of CME on performance.

5.1 Reduction in Hamming Weight

We evaluate the hamming weight reduction when using Bit Plane Compression with and without hamming-weight-aware inversion scheme and compare it against a previously proposed Dynamic-XOR scheme [47] where the goal was to solely minimize the weight of each cache line. We also included another popularly used cache line compression scheme (BΔI) [31] for this hamming weight analysis. From Figure 11 it can be seen that for all applications both the compression schemes and the Dynamic-XOR scheme reduce hamming weight of cache line as compared to the original weight. BPC without inversion reduces hamming weight by upto 67.3% (avg. 21.3%) compared to the original weight. Adding inversion reduces the hamming weight further (on an avg. 30% compared to the original weight). For most applications, cache line after BPC with or without inversion ends up with a lower hamming weight than

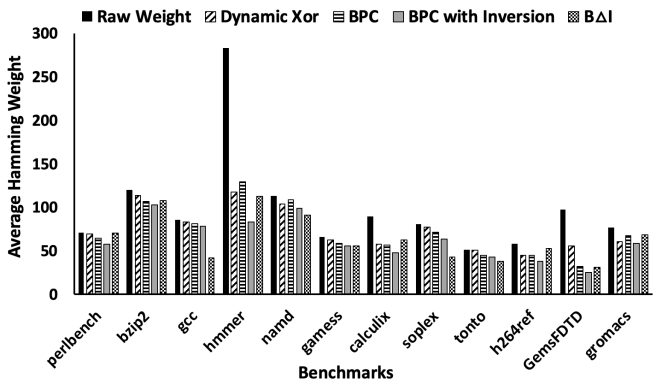

Figure 11: Comparison of average Hamming weight of original cache line, BPC, BΔI and DBX schemes

Dynamic XOR. On an average BPC with inversion has 16.8% lower weight than the Dynamic-XOR scheme. Thus, BPC not only has the advantage of reducing cache line size over Dynamic-XOR which, in turn, allows for stronger ECC, it also reduces hamming weight of the entire cache line, thus reducing chances of unwanted bit flips during write and read operations in STT-RAMs. On top of BPC, hamming weight aware inversion coding further reduces hamming weight by an additional ~8%. BΔI, on an average, performs better than BPC with inversion in the matter of reducing hamming weight of cache lines of most applications. BPC, however, outperforms BΔI in reducing block failure probability and has been discussed in detail in Section 6.1. In a recent work [45], the authors propose to use stronger ECC for cache lines whose hamming weight is above a certain threshold and use weaker ECC otherwise. For this work, the stronger ECC scheme is (72,64) SECDED on each 64-bit cache line word while the weaker ECC is (523,512)SECDED on the entire cache line. In our baseline case, every cache line, irrespective of its hamming weight, gets the stronger protection used in [45] and with CME, the cache lines gets even stronger protection.

5.2 Reduction in block failure probability

To evaluate the reduction in block errors, block failure probability is computed per application for all words in all cache lines retrieved from the memory traces for the following three cases:

- *Baseline:* No Compression and each 64-bit cache line word gets (72,64)SECDED protection.
- *Scheme-1:* Compression (with and without hamming-weight-aware inversion) with Multi-ECC protection scheme where the tag bits are separate.
- *Scheme-2:* Compression (with and without hamming-weight-aware inversion) with Multi-ECC protection scheme where the tag bits are embedded into the cache line.

. As mentioned before, both schemes were compared for two design points with different read disturb/write/retention error rates. The results are shown in Figures 12 and 13. Please note that the y-axes are in logarithmic scale (reverse order). This means that the taller the bar, the smaller is the block failure probability (better it is). Also in both CME schemes, every in-compressible cache line gets the baseline (72,64) SECDED protection per word.

The Compression (with and without inversion) with Multi-ECC (CME) protection scheme is compared against the baseline case where each 64-bit cache line word gets (72,64) SECDED protection. Compared to the baseline, Scheme-1 has 8 extra bits per cache line and Scheme-2 has 2 extra bits. The compression scheme used is the modified version of BPC explained in Section 3.2. We first analyzed the improvement in block failure probability that comes only because of hamming weight reduction by BPC. To do that we computed the reduction in block failure probability when there is BPC alone. The compressed cache lines were padded with zero to increase the final size to 512 and then every 64-bit chunk was provided with (72,64) SECDED protection. For the first design point, BPC with (72,64) SECDED alone (without inversion and multi-bit error correction) reduced block failure probability by at most 4x (average 1.38x). Similarly, for the second design point, the maximum improvement was 4.61x (average 1.35x). These improvements are negligible.

The improvements in block failure probability come mostly from the opportunistic stronger protection that is added to the cache lines. For the first design point, BPC with Multi-ECC (without inversion) reduces block failure probability by as much as 176x (average 6.18x) and 150x (average 5.11x) for Scheme-1 and Scheme-2 respectively when compared to the baseline. For the second design point, BPC with Multi-ECC (without inversion) reduces block failure probability by as much as 240x (average 6.81x) and 148x (average 5x) for Scheme-1 and Scheme-2 respectively. In fact, successful compression and, therefore, stronger protection is possible for 70.5% cache lines across the benchmarks used. As a result, CME performs dramatically better than the baseline case as well as the case with only compression and SECDED protection. For the baseline case, the benchmark with the worst block failure probability is hmmer. If the entire suite is considered, the reliability of the system would be limited by this benchmark for both design points. With Compression and Multi-ECC, this benchmark, however, has the highest compression ratio and the maximum reduction in the average hamming weight of cache lines. Therefore, it ends up with the maximum reduction in block failure probability. The two benchmarks with the lowest reductions are namd and libquantum. This is because these two benchmarks are the least compressible and hence, most of the cache lines end up with the same (72,64) SECDED protection as the baseline. Across the entire SPEC suite, the maximum block failure probability with the baseline protection scheme is reduced by 5x and 6x when CME Scheme-1 is used for the two design points respectively.

An unexpected result is observed when hamming-weight-aware inversion is used along with BPC. Since inversion helps to reduce the hamming weight of the cache lines, it is expected to reduce the block failure probability beyond what compression without inversion and multi-ecc can achieve. However, for both design points and both schemes it can be seen from the figures that across majority of the benchmarks, the block failure probability is higher with inversion than what it is without inversion. With inversion, CME reduces block failure probability by at most 35x (average 3.2x) compared to the baseline for the first design point (using Multi-ECC Scheme-1). This is significantly lower than the corresponding 6.18x

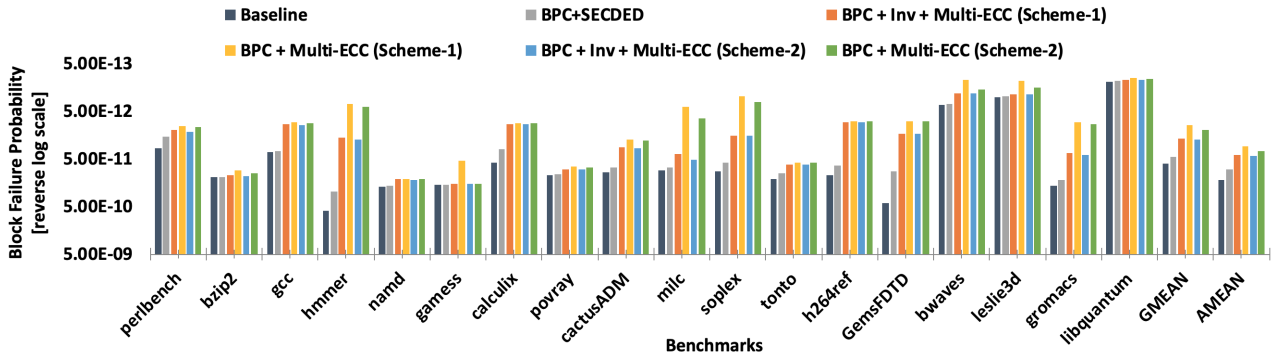

Figure 12: Reduction in block failure probability induced due to write/read/retention errors for the first design point [46] is shown. The y-axis is in logarithmic scale (reverse order). The geometric mean and arithmetic mean of the improvement of CME Schemes over baseline is shown in plot.

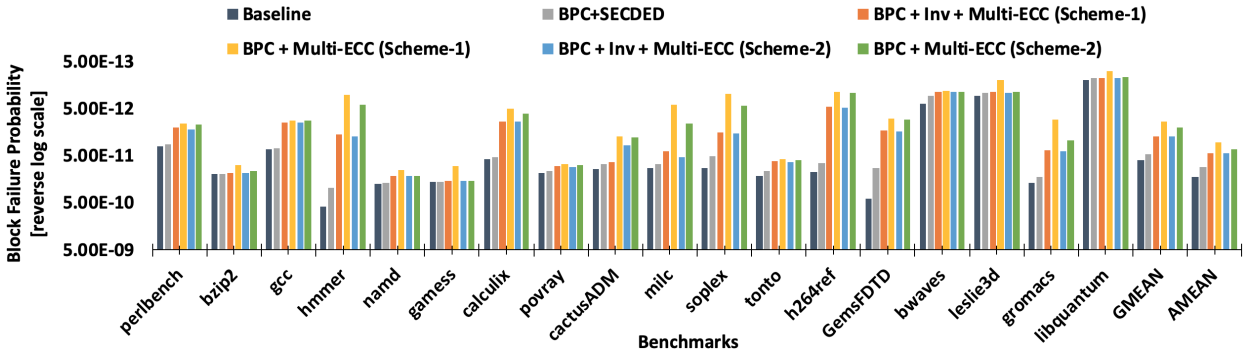

Figure 13: Reduction in block failure probability induced due to write/read/retention errors for the second design point [22] is shown. The y-axis is in logarithmic scale (reverse order). The geometric mean and arithmetic mean of the improvement of CME Schemes over baseline is shown in plot.

average improvement that was achieved without inversion. Though inversion reduces average hamming weight across all benchmarks (see Figure 11), it adds extra bits to the cache lines because each BPC word requires one extra bit to know if the word has been inverted. This can add as many as 35 extra bits to a cache line. As a result, a lot of cache lines end up getting weaker protection. To better utilize the reduction in hamming weight without increasing the cache line size significantly, hamming-weight-aware inversion can be done on groups of multiple BPC words instead of doing inversion on each BPC word. We analyze one such case for design point 2 using Multi-ECC Scheme-1. It is seen that when groups of 3 BPC words are used, BPC with inversion and Multi-ECC (average reduction in block failure probability is 6.14x) outperforms the case without inversion for multiple benchmarks.

To compare against single strong code scheme such as FrugalECC [21], we further evaluate the case of having one stronger code (72,57) DECTED for all cache lines that could be compressed, irrespective of their final size, along with (72,64) SECDED for the uncompressed ones and compared this against CME Schemes. It is seen in Figure 14 that FrugalECC like scheme can achieve at most 34x reduction in block failure probability compared to the baseline

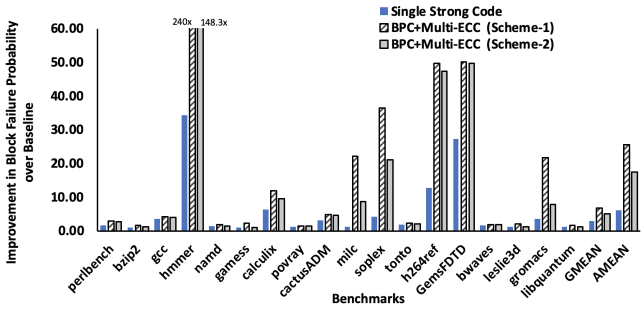

Figure 14: Improvement of CME Schemes 1 and 2 over a scheme that provides uniform (72,57) DECTED for all compressible cache lines and (72,64) SECDED if in-compressible.

while CME Schemes can achieve at much as 240x reduction. For all benchmarks, CME Schemes-1 and 2 perform better than the single strong code scheme.

5.3 Hardware Overhead of Multi-ECC Scheme

The CME scheme requires support for multiple ECC engines (SECDED, DECTED and 3EC4ED). Having multiple ECC encoders and decoders on a memory controller on chip can be costly in terms of both area as well as power. However, if asymmetric quantum BCH coding [2] is used, G and H matrices for a smaller ECC (for e.g., DECTED) can be composed out of sub-matrices of G and H matrices of a stronger ECC scheme (for eg. 3EC4ED), therefore the same hardware can be reused. Since in our case, the total codeword length is same for all the cases (n=72 bits), eliminating rows from the bottom of the H-matrix of a stronger code (for e.g., 3EC4ED) would generate the H-matrix of a weaker code (for e.g., DECTED) with the same codeword length. However, for encoding using G-matrix, the rows (=k) of the G-matrix decrease as we move towards a stronger ECC code for a given constant codeword length (=n). Also, (72,57) DECTED H-matrix can re-use a (144,127) DECTED H-matrix. Thus, the largest decoder would be required for the (144,127) DECTED and the other BCH based codes ((72,50) 3EC4ED and (72,57) DECTED) can re-use most of the decoder. (72,64) SECDED code would require a separate decoder. However, even the baseline (72,64) SECDED protection scheme would require the same parity check engine. Synthesizing the parity check engines of (144, 127) DECTED using an industrial 45nm library results in about only about 8700 μm^2 of additional area overhead compared to a only SECDED implementation as in the baseline. Moreover, since only one word is decoded at a time, reuse isn't expected to have any performance degradation.

As reported in [20], the area overhead of BPC compression engine is about 48000 μm^2 when synthesized using 40nm TSMC standard cells. The total area overhead of BPC and ECC encoding/decoding engines is not significant when compared to the currently available large sizes of processors. The per-bit energy overhead of BPC is less than 1% of per-bit STT-RAM read energy [43]. The latency overhead of CME and the possible impact of system performance is evaluated next.

5.4 System Performance Evaluation

As mentioned in [20], Bit-Plane compression takes 7 cycles for compression/decompression. Since in our case each DBX word is less than half their size, we should be able to fit in twice the number of DBX encoders in the same area. As a result, we would be able to complete the encoding step (last step in BPC) in 2 cycles instead of 4. Thus, we would require 5 cycles for BPC compression/decompression. The latency overhead of fetching the tag bits from the memory would be different for the two CME Schemes.

In Scheme-1, each bit of tag is fetched in each cycle. As discussed earlier ECC decoding can begin after 1 cycle if the cache line is not compressed and after 4 cycles when compressed (only after all 4 bits of tag are read)[1]. In our performance simulations we have conservatively assumed that every cache line is compressed and thus, would incur a latency overhead of 4 cycles every time a cache line is read from the memory when using CME Scheme 1 as compared to the baseline. From our synthesis results we see that even the largest ECC decoding gets done in one cycle. Thus, for stronger protection there is no additional latency overhead. However, in many of today's memory systems, the payload word gets read from the memory in the first burst even when it is not the first cache line word and the rest of the cache line words get read in the successive bursts. This is called priority word first [26] and is done to improve performance. In our case, we have to wait for the entire cache line to arrive before we can start with the decompression. Thus, priority word first cannot be implemented here. To account for this we have taken an average of 4 cycle overhead. Thus, overall CME Scheme 1 has 13 cycle latency overhead as compared to baseline.

For Scheme 2 the BPC overhead remains the same as Scheme-1. In this scheme, both bits of tag are read in one burst. Therefore, the additional latency overhead for fetching tag is 1 cycle. Here also the priority word first cannot be implemented and thus, additional 4 cycles are taken. Overall accessing a cache line in the memory in Scheme-2 will take 10 more cycles compared to the baseline. The performance results are shown in Figure 15. For the system with 8 InO cores, the performance degradation for Scheme-1 is, at most, $\sim$ 6.9% (avg. 1.8%) and for Scheme-2 is, at most $\sim$ 3% (avg. 1.05%). This 8 core system had only 2MB shared Last Level Cache (LLC) and had minimal memory access latency hiding techniques like prefetching enabled. As a result, this exaggerates the latency overhead of CME. For the system with a single OoO core and 2MB LLC, the performance degradation when using Scheme-1 is, at most, $\sim$ 4.5% (avg. 1.6%) and, for Scheme-2 is, at most, $\sim$ 3.3% (avg. 1.1%). As expected, for OoO core with a larger cache and no competition among cores for cache space and better memory access latency hiding techniques like prefetching, the performance impact when using CME over the baseline is lesser than the previous system.

6 DISCUSSION

6.1 Using an Alternative Compression Scheme

The results presented in Section 5.2 were generated using the modified BPC compression scheme discussed in detail in Section 3.2. However, we also analyzed another commonly used cache line compression scheme - BΔI. The results are shown in Figure 16.

The observation was that BΔI with (72,64)SECDED protection on each block performs better than BPC with (72,64) SECDED for majority of benchmarks. This is because, for most of the benchmarks, cache lines have lower hamming weight when compressed with BΔI as compared to BPC. Since the write and read disturb errors in STT-RAM are asymmetric in nature and all cache lines get the same baseline protection irrespective of how much they are compressed, the reduction in block failure probability comes solely from the lowering of hamming weight. Hence, BΔI performs better than BPC. However, when the extra space is opportunistically used for stronger ECC protection, BPC outperforms BΔI for all 18 benchmarks. With BPC and Multi-ECC (Scheme-1), reduction in block failure probability is upto 175x (geomean of 6.18x). But with BΔI and Multi-ECC (Scheme-1), reduction in block failure probability is upto 19.55X (geomean of 2.49x). This is because BPC has better compression ratio than BΔI. This allows most cache lines to get stronger ECC protection. Thus, the compression scheme with the highest average compression ratio needs to be chosen for CME even if the scheme results in higher average hamming weight of cache lines.

[1]Since the tag bits are encoded using systematic SECDED code, the original message, i.e., the 4 bits of tag remain unchanged and the remaining 4 bits of redundancy protecting the tag bits are appended at the end.

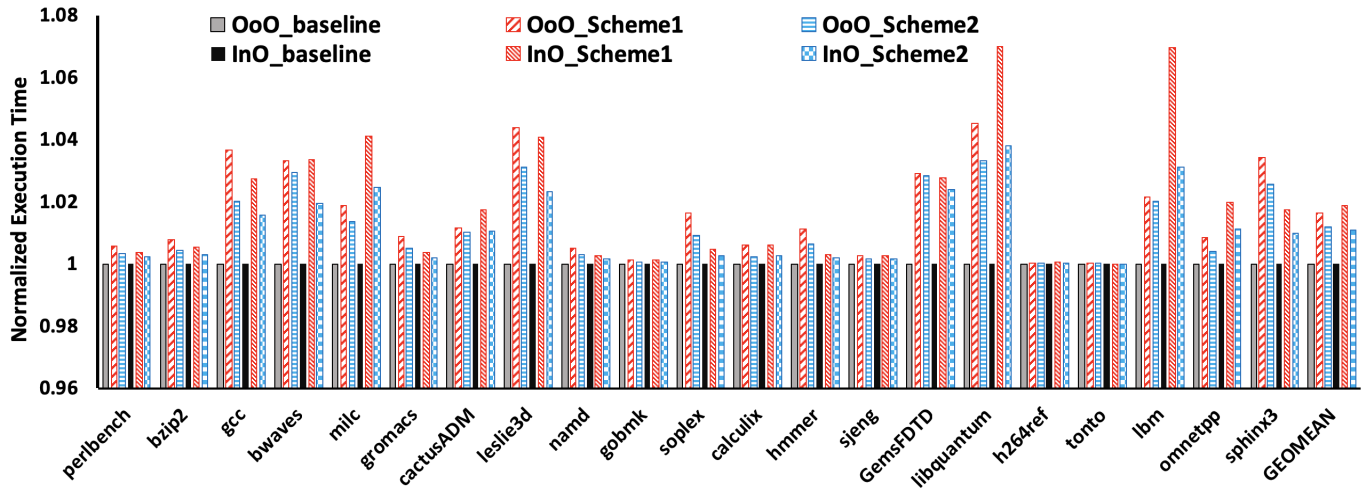

Figure 15: Comparing Normalized Execution Time of two systems (one with 8 InO cores and another with a single OoO core), both having three protection schemes: baseline (72,64)SECDED, CME Scheme-1 and CME Scheme-2. InO and OoO results are normalized to their respective baselines.

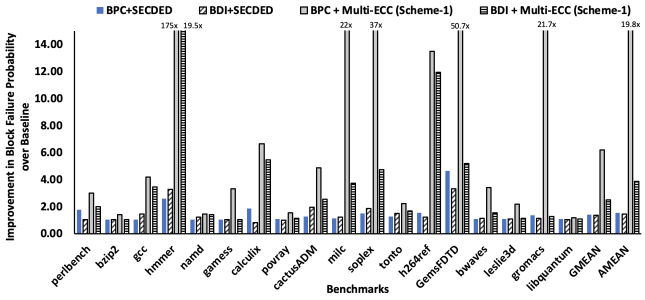

Figure 16: Improvement in Block Failure Probability of BΔI and BPC over Baseline [no compression and (72,64)SECDED].

6.2 Variable Scrubbing Interval

Scrubbing is done in today's systems [16] to reduce the probability of multi-bit errors. As mentioned previously, a refresh operation for STT-RAM would need to be accompanied by an ECC check, which resembles scrubbing where a cache line need to be read in to the memory controller which contains the ECC engine and then written back. Therefore, beyond unavailability of the bank/array, a scrub operation would also consume memory bandwidth. Thus, it is also important to minimize the bandwidth consumption overhead of scrubbing. Based on our analysis we see that for most applications, CME allows to relax the scrubbing interval by as much as 50x as compared to the baseline (72,64)SECDED protection scheme. Another observation is that the scrubbing interval required to achieve a target block failure probability varies among applications. It depends on the compressibility of the cache lines. If the compression ratio of the cache lines is high, the scrubbing interval can be relaxed. For CME, the memory controller needs to check the final compressed cache line size to determine the ECC scheme to be used. This information can be used to provide support for variable scrubbing interval. The system initially starts with the lowest scrubbing interval (maximum scrubbing frequency). A counter keeps track of how many times a cache line gets the strongest ECC protection. If it is beyond a certain threshold, the scrubbing interval can is increased. The counter is reset after a certain period of time or every time the scrubbing interval increases.

6.3 Using STT-RAM as non-ECC DRAM Alternative - Reliability Point of View

In this work we considered the STT-RAM based memory subsystem as main memory with DDR protocol. As a result we compared the MTTF of STT-RAM devices with baseline protection as well as CME protection against that of non-ECC DRAM. Most of the mobile and low power devices using DRAM do not have ECC protection. STT-RAM, because of its power, density and non-volatility benefits over DRAM [24], is considered as a possible DRAM alternative in these devices. However, using STT-RAM memory as a DRAM alternative in such devices would require the STT-RAM device FIT rates to be comparable to what is seen in today's commodity non-ECC DRAM devices. While STT-RAM is not susceptible to radiation induced soft errors [42], the transient read disturb/write/retention error rates can be much worse than the transient bit error rates of DRAM. We use the Samsung design point for our analysis since it has higher bit error rates and used the geometric mean of the block failure probabilities across all benchmarks. We analyze using STT-RAM scrubbing (refresh with ECC) intervals of 64ms (same as the DRAM refresh interval) as well as one second. The DRAM error FIT rate was obtained from [34]. We only considered DRAM transient FIT for our analysis. Note that the FIT rates are for DDR2 DRAM technology. With scaling, the FIT rates for the later DRAM technologies, DDR3/4, are expected to be worse. The results are shown in Figure 17.

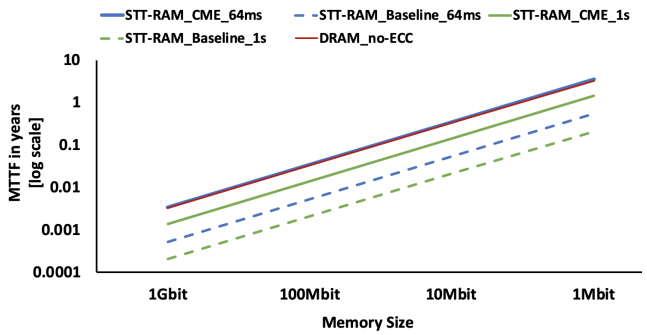

Figure 17: MTTF of STT-RAM devices (with different protection schemes and scrubbing intervals) and non-ECC DRAM devices of different sizes. Note that the y-axis is in log scale.

With baseline protection, the MTTF of STT-RAM device is much lower than a same sized DRAM device, even with 64ms scrubbing interval. In fact, CME with one second scrubbing interval has 2.65x higher MTTF than SECDED based baseline protection with 64ms scrubbing interval. However, it is still not as good as DRAM. With scrubbing interval of 64ms, STT-RAM with CME protection achieves almost similar (*sim*1.07x higher) MTTF as DRAM. Without scrubbing, the MTTF per Mbit, with CME, drops from a few years to less than an hour, making it almost unusable. Thus, CME protection scheme, with scrubbing, allows STT-RAM to be as reliable an alternative as non-ECC DRAM. The baseline (72,64)SECDED protection scheme requires much lower scrubbing interval (less than 6ms) as compared to 64ms with CME, to achieve similar MTTF and thus, makes it almost infeasible energy and performance-wise.

7 CONCLUSION

In this work, we proposed a new ECC protection scheme for STT-RAM based main memories, compression with multi-ECC (CME). First we try to compress every cache line to reduce its size and then apply hamming weight aware inversion coding to reduce the hamming weight of each block. Based on the amount of compression possible, we use the saved additional bits to increase the protection using stronger ECC codes if possible. Compression with inversion itself reduces the hamming weight of the cache lines, thus reducing the probability of 1→0 bit-flips. Opportunistically using stronger ECC codes further helps tolerate multiple bit-flips in a cache line. Our results show that for STT-RAM based main memories, CME can reduce the block failure probability by up to 240x (average 7x) over using a (72,64) SECDED for each cache line word when using two different CME schemes proposed in this paper. The latency and area overheads of CME is minimal with average performance degradation of less than 1.4%.

ACKNOWLEDGMENTS

The authors thank the anonymous MEMSYS'19 program committee and reviewers for their detailed and constructive feedback. The authors thank Dr. Michael B Sullivan from Nvidia Research for helpful discussions and his feedback on this work.

REFERENCES

[1] A. R. Alameldeen and D. A. Wood. 2004. Adaptive cache compression for high-performance processors. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.* 212–223. https://doi.org/10.1109/ISCA.2004.1310776
[2] Salah Aly Ahmed. 2008. Asymmetric and Symmetric Subsystem BCH Codes and Beyond. (04 2008).
[3] B. Del Bel, J. Kim, C. H. Kim, and S. S. Sapatnekar. 2014. Improving STT-MRAM density through multibit error correction. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE).* 1–6. https://doi.org/10.7873/DATE.2014.195
[4] R. A. Bheda, J. A. Poovey, J. G. Beu, and T. M. Conte. 2011. Energy efficient Phase Change Memory based main memory for future high performance systems. In *2011 International Green Computing Conference and Workshops.* 1–8. https://doi.org/10.1109/IGCC.2011.6008569
[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718
[6] R. Bishnoi, M. Ebrahimi, F. Oboril, and M. B. Tahoori. 2014. Read disturb fault detection in STT-MRAM. In *2014 International Test Conference.* 1–7. https://doi.org/10.1109/TEST.2014.7035342
[7] Carl Bracken and Tor Helleseth. 2009. Triple-Error-Correcting BCH-Like Codes. *CoRR* abs/0901.1827 (2009). arXiv:0901.1827 http://arxiv.org/abs/0901.1827
[8] C. L. Chen and M. Y. Hsiao. 1984. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. *IBM Journal of Research and Development* 28, 2 (March 1984), 124–134. https://doi.org/10.1147/rd.282.0124
[9] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. A. Wolf, A. W. Ghosh, J. W. Lu, S. J. Poon, M. Stan, W. H. Butler, S. Gupta, C. K. A. Mewes, T. Mewes, and P. B. Visscher. 2010. Advances and Future Prospects of Spin-Transfer Torque Random Access Memory. *IEEE Transactions on Magnetics* 46, 6 (June 2010), 1873–1878. https://doi.org/10.1109/TMAG.2010.2042041
[10] L. Chen, Y. Cao, and Z. Zhang. 2013. Free ECC: An efficient error protection for compressed last-level caches. In *2013 IEEE 31st International Conference on Computer Design (ICCD).* 278–285. https://doi.org/10.1109/ICCD.2013.6657054
[11] Zhitao Diao, Zhanjie Li, Shengyuang Wang, Yunfei Ding, Alex Panchula, Eugene Chen, Lien-Chang Wang, and Yiming Huai. 2007. Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *J. Phys.: Condens. Matter* 19 (04 2007), 165209–13. https://doi.org/10.1088/0953-8984/19/16/165209
[12] Y. Emre, C. Yang, K. Sutaria, Y. Cao, and C. Chakrabarti. 2012. Enhancing the Reliability of STT-RAM through Circuit and System Level Techniques. In *2012 IEEE Workshop on Signal Processing Systems.* 125–130. https://doi.org/10.1109/SiPS.2012.11
[13] Everspin. 2019. Embedded MRAM. https://www.everspin.com/everspin-embedded-mram
[14] M. Y. Hsiao. 1970. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. *IBM Journal of Research and Development* 14, 4 (July 1970), 395–401.
[15] Intel. 2013. Memory Resiliency. *Technology Journal* 17, 1 (May 2013).
[16] Bruce Jacob, Spencer W. Ng, and David T. Wang. 2008. *Memory Systems: Cache, DRAM, Disk.*
[17] Myoungsoo Jung, Youngbin Jin, and Mustafa Shihab. 2014. Area, Power and Latency Considerations of STT-MRAM to Substitute for Main Memory. In *The Memory Forum.*
[18] S. Kaneda and E. Fujiwara. 1982. Single Byte Error Correcting Double Byte Error Detecting Codes for Memory Systems. *IEEE Trans. Comput.* 31, 7 (1982), 596–602.
[19] T. Kawahara, R. Takemura, K. Miura, J. Hayakawa, S. Ikeda, Y. Lee, R. Sasaki, Y. Goto, K. Ito, T. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2007. 2Mb Spin-Transfer Torque RAM (SPRAM) with Bit-by-Bit Bidirectional Current Write and Parallelizing-Direction Current Read. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers.* 480–617. https://doi.org/10.1109/ISSCC.2007.373503
[20] J. Kim, M. Sullivan, E. Choukse, and M. Erez. 2016. Bit-Plane Compression: Transforming Data for Better Compression in Many-Core Architectures. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA).* 329–340.
[21] Jungrae Kim, Michael Sullivan, Seong-Lyong Gong, and Mattan Erez. 2015. Frugal ECC: Efficient and Versatile Memory Error Protection Through Fine-grained Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15).* ACM, New York, NY, USA, Article 12, 12 pages. https://doi.org/10.1145/2807591.2807659
[22] J. H. Kim, W. C. Lim, U. H. Pi, J. M. Lee, W. K. Kim, J. H. Kim, K. W. Kim, Y. S. Park, S. H. Park, M. A. Kang, Y. H. Kim, W. J. Kim, S. Y. Kim, J. H. Park, S. C. Lee, Y. J. Lee, J. M. Yoon, S. C. Oh, S. O. Park, S. Jeong, S. W. Nam, H. K. Kang, and E. S. Jung. 2014. Verification on the extreme scalability of STT-MRAM without loss of thermal stability below 15 nm MTJ cell. In *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers.* 1–2. https://doi.org/10.1109/VLSIT.2014.6894366

[23] N. Kim and K. Choi. 2015. A design guideline for volatile STT-RAM with ECC and scrubbing. In *2015 International SoC Design Conference (ISOCC)*. 29–30. https://doi.org/10.1109/ISOCC.2015.7401649

[24] Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. *ISPASS 2013 - IEEE International Symposium on Performance Analysis of Systems and Software*, 256–267. https://doi.org/10.1109/ISPASS.2013.6557176

[25] H. Li, X. Wang, Z. Ong, W. Wong, Y. Zhang, P. Wang, and Y. Chen. 2011. Performance, Power, and Reliability Tradeoffs of STT-RAM Cell Subject to Architecture-Level Requirement. *IEEE Transactions on Magnetics* 47, 10 (Oct 2011), 2356–2359. https://doi.org/10.1109/TMAG.2011.2159262

[26] Micron. 2019. DDR4 SDRAM. https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf.

[27] S. Mittal and J. S. Vetter. 2016. A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 5 (May 2016), 1537–1550.

[28] S. Mittal, J. S. Vetter, and L. Jiang. 2017. Addressing Read-disturbance Issue in STT-RAM by Data Compression and Selective Duplication. *IEEE Computer Architecture Letters* PP, 99 (2017), 1–1.

[29] H. Noguchi, K. Kushida, K. Ikegami, K. Abe, E. Kitagawa, S. Kashiwada, C. Kamata, A. Kawasumi, H. Hara, and S. Fujita. 2013. A 250-MHz 256b-I/O 1-Mb STT-MRAM with advanced perpendicular MTJ based dual cell for nonvolatile magnetic caches to reduce active power of processors. In *2013 Symposium on VLSI Technology*. C108–C109.

[30] D. J. Palframan, N. S. Kim, and M. H. Lipasti. 2015. COP: To compress and protect main memory. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 682–693. https://doi.org/10.1145/2749469.2750377

[31] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry. 2012. Base-delta-immediate compression: Practical data compression for on-chip caches. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 377–388.

[32] A. Raychowdhury. 2013. Pulsed READ in spin transfer torque (STT) memory bitcell for lower READ disturb. In *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 34–35. https://doi.org/10.1109/NanoArch.2013.6623037

[33] Vijay Sathish, Michael J. Schulte, and Nam Sung Kim. 2012. Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*. ACM, New York, NY, USA, 325–334. https://doi.org/10.1145/2370816.2370864

[34] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM Errors in the Wild: A Large-scale Field Study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*. ACM, New York, NY, USA, 193–204. https://doi.org/10.1145/1555349.1555372

[35] Nak Hee Seong, Sungkap Yeo, and Hsien-Hsin S. Lee. 2013. Tri-level-cell Phase Change Memory: Toward an Efficient and Reliable Memory System. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. 440–451.

[36] D. Shum, D. Houssameddine, S. T. Woo, Y. S. You, J. Wong, K. W. Wong, C. C. Wang, K. H. Lee, K. Yamane, V. B. Naik, C. S. Seet, T. Tahmasebi, C. Hai, H. W. Yang, N. Thiyagarajah, R. Chao, J. W. Ting, N. L. Chung, T. Ling, T. H. Chan, S. Y. Siah, R. Nair, S. Deshpande, R. Whig, K. Nagel, S. Aggarwal, M. DeHerrera, J. Janesky, M. Lin, H. J. Chia, M. Hossain, H. Lu, S. Ikegawa, F. B. Mancoff, G. Shimon, J. M. Slaughter, J. J. Sun, M. Tran, S. M. Alam, and T. Andre. 2017. CMOS-embedded STT-MRAM arrays in 2x nm nodes for GP-MCU applications. In *2017 Symposium on VLSI Technology*. T208–T209. https://doi.org/10.23919/VLSIT.2017.7998174

[37] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy. 2015. Challenges for High-Density 16Gb ReRAM with 27nm Technology. In *Symposium on VLSI Circuits (VLSI Circuits)*. T106–T107.

[38] M. R. Stan and W. P. Burleson. 1995. Bus-invert coding for low-power I/O. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 3, 1 (March 1995), 49–58. https://doi.org/10.1109/92.365453

[39] Suock Chung, K. Rho, S. Kim, H. Suh, D. Kim, H. Kim, S. Lee, J. Park, H. Hwang, S. Hwang, J. Lee, Y. An, J. Yi, Y. Seo, D. Jung, M. Lee, S. Cho, J. Kim, G. Park, Gyuan Jin, A. Driskill-Smith, V. Nikitin, A. Ong, X. Tang, Yongki Kim, J. Rho, S. Park, S. Chung, J. Jeong, and S. Hong. 2010. Fully integrated 54nm STT-RAM with the smallest bit cell dimension for high density memory application. In *2010 International Electron Devices Meeting*. 12.7.1–12.7.4. https://doi.org/10.1109/IEDM.2010.5703351

[40] R. Takemura, T. Kawahara, K. Ono, K. Miura, H. Matsuoka, and H. Ohno. 2010. Highly-scalable disruptive reading scheme for Gb-scale SPRAM and beyond. In *2010 IEEE International Memory Workshop*. 1–2. https://doi.org/10.1109/IMW.2010.5488324

[41] M. Thuresson, L. Spracklen, and P. Stenstrom. 2008. Memory-Link Compression Schemes: A Value Locality Perspective. *IEEE Trans. Comput.* 57, 7 (July 2008), 916–927. https://doi.org/10.1109/TC.2008.28

[42] S. Wang, H. C. Hu, H. Zheng, and P. Gupta. 2016. MEMRES: A Fast Memory System Reliability Simulator. *IEEE Transactions on Reliability* 65, 4 (Dec 2016), 1783–1797.

[43] S. Wang, H. Lee, F. Ebrahimi, P. K. Amiri, K. L. Wang, and P. Gupta. 2016. Comparative Evaluation of Spin-Transfer-Torque and Magnetoelectric Random Access Memory. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 2 (June 2016), 134–145. https://doi.org/10.1109/JETCAS.2016.2547681

[44] S. Wang, A. Pan, C. O. Chui, and P. Gupta. 2017. Tunneling Negative Differential Resistance-Assisted STT-RAM for Efficient Read and Write Operations. *IEEE Transactions on Electron Devices* 64, 1 (Jan 2017), 121–129. https://doi.org/10.1109/TED.2016.2631544

[45] X. Wang, M. Mao, E. Eken, W. Wen, H. Li, and Y. Chen. 2016. Sliding Basket: An adaptive ECC scheme for runtime write failure suppression of STT-RAM cache. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 762–767.

[46] L. Wei, J. G. Alzate, U. Arslan, J. Brockman, N. Das, K. Fischer, T. Ghani, O. Golonzka, P. Hentges, R. Jahan, P. Jain, B. Lin, M. Meterelliyoz, J. OâĂŹDonnell, C. Puls, P. Quintero, T. Sahu, M. Sekhar, A. Vangapaty, C. Wiegand, and F. Hamzaoglu. 2019. 13.3 A 7Mb STT-MRAM in 22FFL FinFET Technology with 4ns Read Sensing Time at 0.9V Using Write-Verify-Write Scheme and Offset-Cancellation Sensing Technique. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 214–216. https://doi.org/10.1109/ISSCC.2019.8662444

[47] W. Wen, M. Mao, X. Zhu, S. H. Kang, D. Wang, and Y. Chen. 2013. CD-ECC: Content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[48] L. Yang, Y. Cheng, Y. Wang, H. Yu, W. Zhao, and A. Todri-Sanial. 2015. A body-biasing of readout circuit for STT-RAM with improved thermal reliability. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1530–1533. https://doi.org/10.1109/ISCAS.2015.7168937

[49] Yiming Huai, M. Pakala, Zhitao Diao, and Yunfei Ding. 2005. Spin-transfer switching current distribution and reduction in magnetic tunneling junction-based structures. *IEEE Transactions on Magnetics* 41, 10 (Oct 2005), 2621–2626. https://doi.org/10.1109/TMAG.2005.855346

[50] Y. Zhang, X. Wang, Y. Li, A. K. Jones, and Y. Chen. 2012. Asymmetry of MTJ switching and its implication to STT-RAM designs. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1313–1318. https://doi.org/10.1109/DATE.2012.6176695

[51] W.S. Zhao, T. Devolder, Y. Lakys, J.O. Klein, C. Chappert, and P. Mazoyer. 2011. Design considerations and strategies for high-reliable STT-MRAM. *Microelectronics Reliability* 51, 9 (2011), 1454 – 1458. https://doi.org/10.1016/j.microrel.2011.07.001 Proceedings of the 22th European Symposium on the RELIABILITY OF ELECTRON DEVICES, FAILURE PHYSICS AND ANALYSIS.