Secure, Reliable, and Energy-efficient Phase Change Main Memory

by

Stephen Longofono

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2021

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Stephen Longofono

It was defended on

July 1, 2021

and approved by

Jingtong Hu, Ph.D., Assistant Professor, Dept. of Electrical and Computer Engineering

Feng Xiong, Ph.D., Assistant Professor, Dept. of Electrical and Computer Engineering

Alex K. Jones, Ph.D., Professor, Dept. of Electrical and Computer Engineering

Secure, Reliable, and Energy-efficient Phase Change Main Memory

Stephen Longofono, M.S.

University of Pittsburgh, 2021

Recent trends in supercomputing, shared cloud computing, and "big data" applications are placing ever-greater demands on computing systems and their memories. Such applications can readily saturate memory bandwidth, and often operate on a working set which exceeds the capacity of modern DRAM packages. Unfortunately, this demand is not matched by DRAM technology development. As Moore's Law slows and Dennard Scaling stops, further density improvements in DRAM and the underlying semiconductor devices are difficult [1]. In anticipation of this limitation, researchers have pursued emerging memory technologies that promise higher density than conventional DRAM devices. One such technology in phase-change memory (PCM) is especially desirable due to its increased density relative to DRAM. However, this nascent memory has outstanding challenges to overcome before it is viable as a DRAM replacement. PCM devices have limited write endurance, and can consume more energy than their DRAM counterparts, necessitating careful control of how and how often they are written. A second challenge is the non-volatile nature of PCM devices; many applications rely on the volatility of DRAM to protect security critical applications and operating system address space between accesses and power cycles. An obvious solution is to encrypt the memory, but the effective randomization of data is at odds with techniques which reduce writes to the underlying memory. This body of work presents three contributions for addressing all challenges simultaneously under the assumption that encryption is required. Using an encryption and encoding technique called CASTLE & TOWERs, PCM can be employed as main memory with up to 30× improvement in device lifetime while opportunistically reducing dynamic energy. A second technique called MACE marries encoding and traditional error-correction schemes providing up to 2.6× improvement in device lifetime alongside a whole-lifetime energy evaluation framework to guide system design. Finally, an architecture called WINDU is presented which supports the application of encoding for an emerging encryption standard with an eye on energy efficiency. Together,

these techniques advance the state-of-the-art, and offer a significant step toward the adoption of PCM as a main memory.

Table of Contents

List of Tables

List of Figures

1.0 Introduction

The demands of "big-data" applications and the limitations of DRAM scaling necessitate the development of denser emerging memory technologies to meet the needs of modern computational workloads [2, 1, 3]. The first devices employing PCM are already coming to market, signaling industry adoption and improving process development for large-scale fabrication. PCM is the most promising replacement for DRAM, given its recent commercial debut in a tiered memory behind conventional DRAM in the Intel Optane Intel Optane Persistent Memory [4]. This combination of DRAM and PCM in the form of 3D X-Point memory [5] bridges the gap between a purely PCM and a purely DRAM main memory to benefit database, big-data analytics, content delivery, and virtual machine infrastructure applications. In addition to presenting a larger working set to such applications, the addition of non-volatile memory in tiered systems allows more rapid reboot for servers and databases, improving response time and performance across power cycles and outage events. However, several challenges to its adoption as a main memory previously identified in the literature remain unsolved—PCM devices have limited write endurance, an asymmetric and relatively higher write energy compared to their DRAM counterparts. In the context of a shared computing environment, the added constraint of data confidentiality defeats techniques designed to reduce energy and exasperates cell wear. These issues persist in the Optane memory modules, as evidenced by the reported per-cell endurance of circa 10^6 writes and the use of AES-XTS encryption to protect private data across power cycles [4, 6]. While it can be expected that the fabrication process will improve reliability and energy consumption with later generations of the technology, these fundamental constraints of cell endurance and data security must be addressed within the memory architecture.

1.1 Contributions & Acknowledgements

My work to date has focused on designs which can be implemented at the memory controller level to reduce writes, mitigate wear-related faults, and opportunistically improve dynamic energy in PCM devices. The contributions of this thesis address all of the above, under the additional constraint that the memory be encrypted. Portions of this work reproduce text and images from my publications "A CASTLE with TOWERs for Reliable, Secure PCM" [7], ©2020 IEEE, and "Toward Secure, Reliable, and Energy Efficient Phase-change Main Memory with MACE" [8], ©2019 IEEE. Specifically, in this thesis I highlight my contributions to these works:

- TOWERs, the first technique to combine compression and the Advanced Encryption Standard (AES) [9] counter-mode encryption in order to find an optimal encoding for data written to endurance-limited memory cells.
- MACE, a combination of encoding and traditional error-correction schemes to improve overall memory lifetime in the context of AES-XTS encryption.
- WINDU, an architecture which supports encoding techniques such as MACE applied in the context of AES-XTS encryption using lightweight compression to reduce overhead.
- LARS, a framework for evaluating the whole-life energy impact of systems using endurance limited memories, in collaboration with my co-author Donald Kline Jr.

The remainder of this thesis is organized as follows: In the following Chapter 2, I provide the necessary background material on PCM technology and its characteristics, error correction schemes for conventional and emerging memories, encoding techniques and their application to memory, encryption schemes based on the AES algorithm, and the pitfalls of combining encryption with encoding. Chapter 3 discusses the CASTLE & TOWERs technique, our simulation framework, our experimental methodology, and the associated results. Chapter 4 discusses the MACE and WINDU techniques, our simulation framework, our experimental methodology, and the associated results. Finally, Chapter 5 presents as summary of these works and my contribution to the state-of-the-art, alongside directions of study for future work.

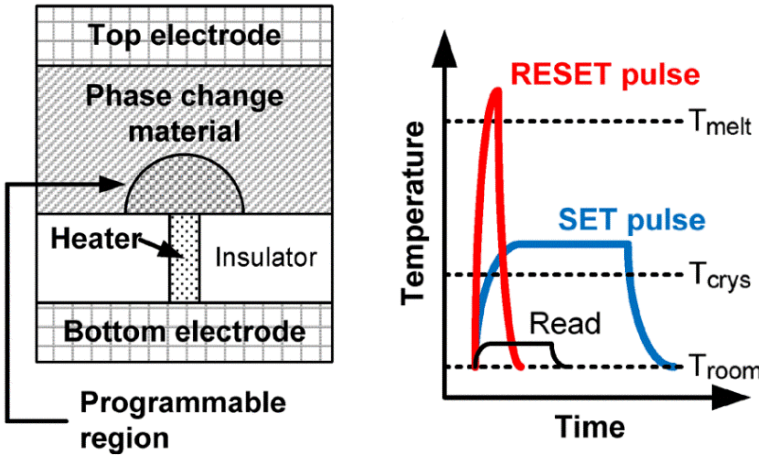
2.0 Background

Endurance limited memories like PCM inevitably encounter endurance faults as their cells approach their nominal lifetime in writes. Beyond basic wear-leveling, various existing techniques developed for error correction and encoding on data channels have been applied to PCM to mitigate these faults. In this chapter, I discuss the relevant characteristics of PCM and its failure modes, alongside existing techniques which have been used to correct errors (transient or otherwise) in conventional memory technologies. Following a discussion of encoding, I present the basic operation of the AES algorithm, and its extensions in Galois-Counter-Mode and AES-XTS. Finally, I present the repercussions of combining error correction and encoding with encryption, motivating the need for techniques which account for both.

2.1 Phase Change Memory

PCM is a resistive memory which encodes data in the phase of the material which composes a PCM cell. Chalcogenides like $Ge_2Sb_2Te_5$ exhibit a relatively high resistance when in an amorphous phase, and a relatively low resistance when in a crystalline lattice phase. Thus an access circuit may interpret the higher resistance phase as logical '0' (RESET) and the lower resistance phase as logical '1' (SET). Figure 1a depicts a typical PCM cell design, wherein the phase change material is embedded in a substrate and is manipulated via a heater and a pair of conducting electrodes. When the potential across the electrodes exceeds a switching threshold (determined by the choice of phase-change material), increased current flows through the cell and induces Joule heating. When the cell exceeds its melting temperature (circa 600^o C) it enters its amorphous state, where it can be held by abruptly removing the potential across the cell. This programs the cell to its RESET state, as indicated by the red pulse in Figure 1b. At the lower crystallization temperature range (circa 100-300^o C), the cell material tends to relax over time into a regular crystalline lattice. In practice, this

is achieved by applying a longer, lower-current SET pulse as depicted in blue in Figure 1b. This interaction of time-under-temperature is an important area of research, since PCM cells have been observed "drifting" toward the crystalline phase in practice at temperatures below the crystallization temperature [10]. Some architecture designs target the drifting problem specifically [11], but in general we must rely on error correction mechanisms to identify and mitigate spurious phase change.



(a) Typical PCM cell architecture. (b) Programming a PCM cell.

Figure 1: A typical PCM cell design and its operation.

The choice of PCM material and cell design is another subject of much research, as it impacts drifting, the speed of state changes, and the overall endurance of individual cells [12]. Novel cell and substrate designs target reduced thermal conductivity [13], and improved switching characteristics [14], which support the scaling and performance goals required to replace DRAM in the long term. However, the inherent limitation on cell endurance persists—individual PCM cells can only be written circa 10^8 times before failing to reliably switch.

A seminal study of PCM reliability classified two permanent failure modes: a failure which results in permanent low-resistance state (SET failure) and a failure which results in permanent high-resistance state (RESET failure) [15]. The former is characterized by a cell remaining in the low-resistance crystalline state in spite of prolonged programming pulses above the switching threshold, indicating that the switching threshold was no longer

permitting adequate current to melt the PCM material. Cells in this state exhibited depletion of Ge and enrichment of Se near the heater element. This "phase-segregation" and field-induced ion migration are indicated as the underlying causes of the SET failure mode [15, 16]. The second fault mode, RESET failure, is characterized by permanent high-resistance across a PCM cell. Physical inspection of such cells reveals a physical separation (void) at the interface of the heater and the phase change material, inhibiting current flow through the cell and exhibiting a higher resistance than both SET and RESET states [15]. Both fault modes become more common as cells age, commonly measured by the number of writes to individual cells. Prototypes in the literature range report a nominal endurance of 10^4-10^{12} writes depending on the material, feature size, and operating conditions, with the most recent commercially available devices rated to 10^6 writes [4].

The consensus among PCM reliability studies to date is that the extrema of temperature accelerates cell wear toward failure, especially the melting temperature applied during a RESET operation. Programming PCM cells requires more time compared to DRAM, representing a longer period of energy transfer to any given cell [17]. With improved process, scaling, and more efficient cell designs, the overall time and energy required to program PCM cells should reduce [14]. In the interim, reducing the writes and in turn energy transfer to the cells can simultaneously improve the efficiency and lifetime of PCM devices. When cells inevitably do fail, we must rely on error detection and correction techniques to prolong the useful lifetime of memories built with PCM.

2.2 Error Correction Techniques

Error correction techniques are widely applied in noisy and unreliable channels to identify and correct spurious changes in the data. Most of these techniques were developed to address transient faults in memories and communication channels, although they can obviously be applied to permanent or otherwise persistent faults in non-volatile memories. The most ubiquitous class of error correction techniques are called error correcting codes (ECC), which compute parity information on a message, and the concatenation of the message and

its parity form a code word. Modern storage devices typically employ Single-error-correction, double-error-detection (SECDED) codes which are based on the overlapping parity set techniques of Hamming codes [18]. A typical SECDED codeword consists of 64 data bits plus eight bits of parity information to match the word size of conventional processors. As the name implies, SECDED is capable of correcting a single bit in error, or detect (but not locate) two bits in error per 72-bit codeword. Especially for non-volatile memories like hard disks and flash, more sophisticated parity encoding schemes have been developed which correct more bits per correction block, including Reed-Solomon codes [19], Bose-Chaudhuri-Hocquenghem codes [20], and Low-density parity codes [21]. All of the above use more complex representations of the underlying message in polynomials, enhancing the correction capabilities at the expense of encoding latency and auxiliary storage associated with storing parity information. Collectively, I will refer to this class of schemes for the remainder of this text by ECCN, denoting an Error Correcting Code which can correct up to N bits in error per correction block. In the context of DRAM and main memory, ECC1 (SECDED Hamming codes) is the gold standard for comparison—commodity DRAM designs which provide ECC already include additional capacity to store parity information, thus ECC1 correction capability and overhead are a natural benchmark for error correction techniques.

Pointer-based error correction techniques leverage the same auxiliary capacity to instead store replacement bits for persistent faults in the correction block, alongside a pointer to the bit address of the faulty location within that block. Error Correcting Pointers [22] operate on groups of 512 bits, storing N pointers of length $log_2(512)$ bits alongside a replacement bit for each known faulty location, for a total of 10 auxiliary bits required for each correctable error per block. ECP is more flexible than ECC1 in that it can handle cases where more than two faults occur within the same correction block, for example if certain rows or columns of memory exhibit spatial correlation of faults due to process variation [23, 24, 25]. However, the fact that ECP must use its own pointers to correct faults in the auxiliary space can quickly degrade the correction capability of the block. Similarly to ECCN, for the remainder of this text I will refer to this technique by ECPN, where N indicates how many pointers (and correctable errors) are associated with each correction block.

An important consideration when mitigating persistent faults is the state of the faulty cell. For memories using DRAM, a faulty cell would either fail to charge or fail to hold its charge—in both cases, reading the cell would yield logical '0' at the sense amplifier. For PCM, depending on which failure mode occurred first, reading a faulty cell could result in logical '1' (SET failure) or logical '0' (RESET failure). In both cases, the fact that these "stuck" cells can still produce a value when read offers an opportunity to improve fault mitigation. If a SET failure occurs, and a '1' is to be written to that cell, then there is no error, and we call that cell "stuck-at-right" (SA-R). If instead a '0' is to be written to that cell, then the fault creates an error, and we call that cell "stuck-at-wrong"(SA-W). In the context of the above error correction schemes, fault tolerance can be improved provided there is a way to identify and track persistent faults as they develop. To this end, several schemes have been introduced in the literature to identify faults and encode them into a fault map [26, 27, 28]. For the remainder of this work, I assume that such a mechanism is in place.

Both of the above error correction schemes are widely employed to mitigate faults in non-volatile memories, but there are more general transformations of data which can be employed to a similar effect. In the next section, I present the concept of coset encoding and its common uses in write reduction and error correction.

2.3 Coset Encoding

Sets of codes collectively referred to as Cosets are used to modify data (typically via `XOR`) to match a desired pattern before writing it to some media. In the context of this work, the most relevant target is reducing writes and masking endurance faults, although these techniques are general enough to target arbitrary optimization goals such as reducing specific data symbols, reducing dynamic energy, or reducing changes of data values. As I will show in Chapters 3 and 4, choosing a coset approach which provides many candidate transformations can evaluate multiple objectives at once; if there are more than one transformation which meet a primary objective such as masking faults, a secondary objective such as energy

reduction can be applied to select the best coset. One of the most simple applications of coset transformation is to invert data. Data Bus Inversion (DBI) [29] was developed to reduce energy associated with data transmission over high-speed bus lines, but the approach is functionally equivalent to reducing writes in a memory. If the data block being written would result in more than half of all cells changing state, then simply inverting the data before writing it reduces changes and in turn wear on the underlying memory.

The block inversion approach belongs to a class of geometric error correction schemes which reinterpret blocks of data as flattened matrices, which are then separated into partitions that can be inverted to mask the effects of persistently faulty cells. Flip-N-Write [30] adapts DBI to memory applications with finer-granularity correction blocks. SAFER [31] uses a similar structure to ECP to point to specific partitions and indicate if they should be inverted. This basic idea is extended by AEGIS [32] to allow more sophisticated partitioning schemes, improving the likelihood that faults will fall in different partitions which in turn improves the likelihood that inverting partitions can mask two or more faults. Of course, there are tradeoffs to consider in how partition membership and inversion are encoded, but depending on the fault tolerance needs of the system, this increased auxiliary overhead may be acceptable.

The aforementioned coset techniques are most effective when data exhibit patterns, *i.e.,* data are *biased* toward specific values. For example, binary representations of integers such as two's complement tend to use less than the full range of the bits allocated, resulting in the most significant bits being all '0' or all '1' most of the time. In this case, simply applying DBI at the granularity of half the word size will effectively reduce writes to upper half of the memory cells in each word.

Unfortunately, in the context of encrypted non-volatile memory, the patterns which make these techniques effective are removed. Private user data, security-critical operating system code, and sensitive authentication tokens might exist in main memory at any given time, necessitating encryption to protect data in the event of power disruption or side-channel attack. One of the primary goals of encryption is data confidentiality: by design, the encrypted version of any message should not reveal any information about the message itself. Thus if we employ encryption, we must adopt coset techniques which account for the apparent

randomness of encrypted data. To this end, the use of random cosets can be effective [33, 34]. Consider the ideal case where every word of length N has an equal probability of '0' or '1' for each bit. Thus in any given word, the probability of exactly x bits changing follows a Bernoulli distribution, which can be computed directly as a special case of the binomial distribution:

$$P(x) = \binom{N}{x} \cdot 0.5^x \cdot 0.5^{N-x} \tag{1}$$

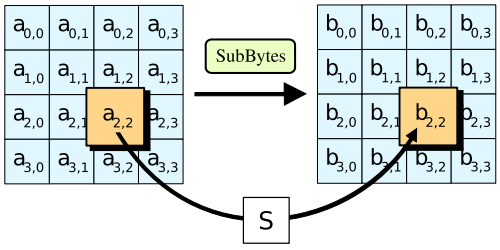
The closed-form expression for the expected value of bit changes is given by $N \cdot p$, where here N represents the length of the data block in bits and $p = 0.5$ represents the probability that any given bit will be changed during a write. Thus on average, we expect an equal number of bits to change in any given word when both the existing and new data are unbiased. This result is the theoretical basis for the poor performance of inversion techniques under effectively randomized data. Previous study demonstrated that for unbiased, effectively random data, using pseudorandom-valued cosets consistently outperforms techniques which target biased data [34]. This motivates the use of encrypted data and other artifacts of the encryption process as cosets in Chapters 3 and 4.

In the next section, the basic concepts of AES encryption and its common applications in AES-GCM and AES-XTS are presented alongside their relative strengths and weaknesses for use with non-volatile memory.

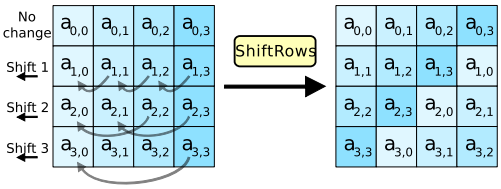
2.4 AES-Based Encryption

The Advanced Encryption Standard (AES) [9] was introduced in 2001 as the standard cipher for cryptography, based on the Rijndael algorithm published in a prior work [35]. AES is a symmetric key algorithm wherein a secret input key (128, 192, or 256 bits long) is distributed to approved parties, which is used along with 128-bit blocks of plaintext data as input to produce 128-bit blocks of encrypted ciphertext as output. The algorithm itself consists of several rounds with four distinct steps which transform each of the 16 bytes in the 128-bit input:

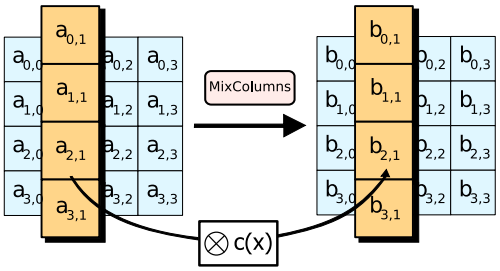
- Substitute Bytes: each byte of input is used to index a lookup table, implementing a nonlinear transformation as shown in Figure 2a.
- Shift Rows: each row i in a 4x4 row-major matrix of the input is barrel shifted by i byte positions as shown in Figure 2b.
- Mix Columns: The 4x4 row-major matrix of the input is multiplied by a fixed transformation matrix to implement an invertible linear transformation on its columns as shown in Figure 2c.
- Add Round Key: The secret key and the round number k is used to generate a round key. This round key is applied via bitwise `XOR` to the input data as shown in Figure 2d.



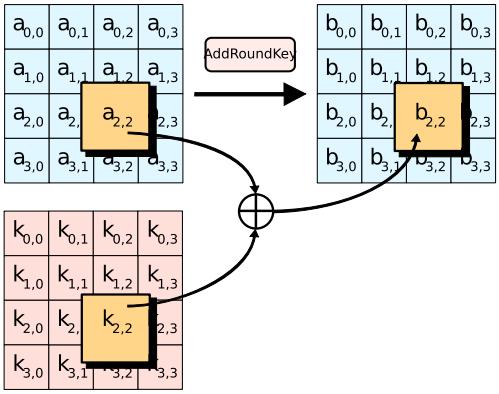
(a) AES substitute bytes step using a pre-generated lookup table.



(b) AES shift rows step.



(c) AES mix columns step via matrix multiply with transform $c(x)$



(d) AES add round key step via `XOR`.

Figure 2: The four transformation steps used in each round of AES encryption.

The various steps are repeated for each of {10, 12, 14} rounds for {128, 192, 256}-bit keys, respectively, where the input of the first round is the plaintext data and the input of subsequent rounds is the output of the previous round. Each step is invertible, thus the

decryption process is the same but with all rounds and steps applied in reverse order, using the secret key and the ciphertext as the initial input. However, to sufficiently disperse data which may span many encryption blocks, AES is commonly used in Cipher Block Chain (CBC) mode.

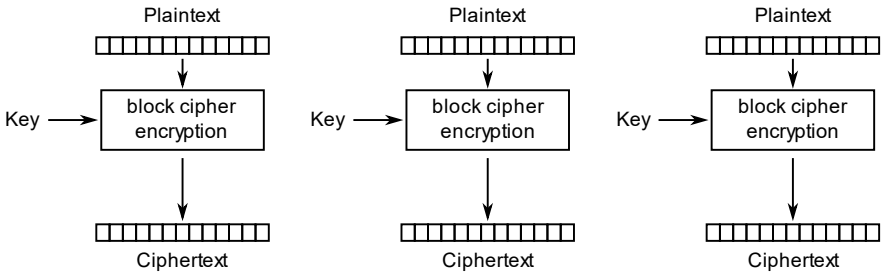


Figure 3: AES Electronic Code Book mode.

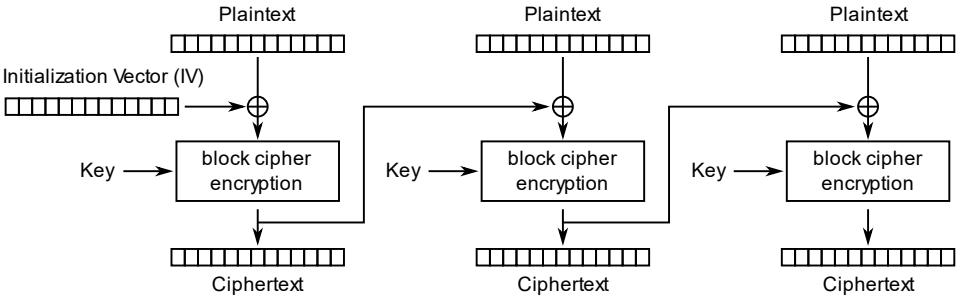


Figure 4: AES Cipher Block Chain mode.

If each block is encrypted independently (commonly referred to as Electronic Code Book (ECB) mode, see Figure 3), patterns in the data are exposed in the output. The reason for this is that for any given plaintext and secret key pair, the output of the AES algorithm is deterministic. Thus if there are patterns in the data which span the length of the 128-bit encryption block size, those patterns will persist as their encrypted ciphertexts will be identical. To remedy this, CBC mode replaces the input to encryption block i by the

ciphertext of block $(i-1)$ XOR'd with the plaintext of block i as shown in Figure 4. With proper padding to a multiple of the 128-bit block size, this ensures that data dispersion is not sensitive to the size of the input data. This approach is well-suited for encrypting streams of data, but limits the amount of parallelization that can be applied. As an alternative for performance-sensitive applications, Galois counter-mode [36](GCM) was developed.

2.4.1 AES in Galois-Counter-Mode

Figure 5 shows the basic operation of AES-GCM. For each encryption block, a counter is maintained, which is concatenated with a nonce value to form the input of the AES algorithm in ECB mode along with the secret key. The output of the AES algorithm is commonly called the one-time-pad (OTP), which is then applied via XOR to the corresponding block of plaintext to produce the ciphertext output. Decryption follows this process in reverse, where the counter value is read to reproduce the OTP, which is then XOR'd with the ciphertext to recover the plaintext.

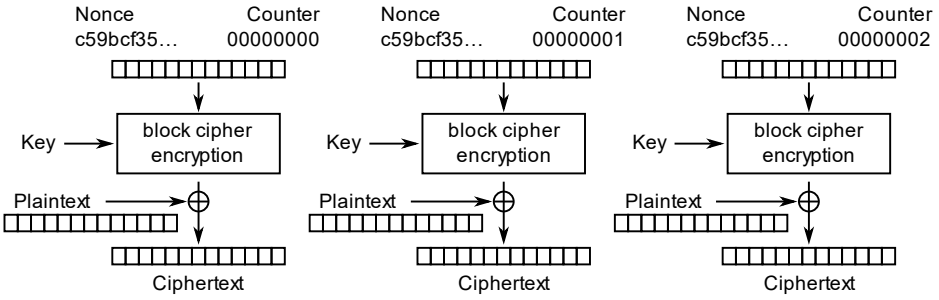


Figure 5: AES Galois counter-mode.

This approach is "embarrassingly parallel," making it desirable for high-bandwidth applications like encrypting memory. However, in order to maintain the security of this approach, it is critical that the nonce values must never be reused. If a malicious actor knew that a nonce value was reused, they can encrypt a known plaintext and greatly reduce the search space for the output of the AES block. Likewise, it is important that the counter values be long enough that they cannot easily be saturated, which is equivalent to reusing a counter.

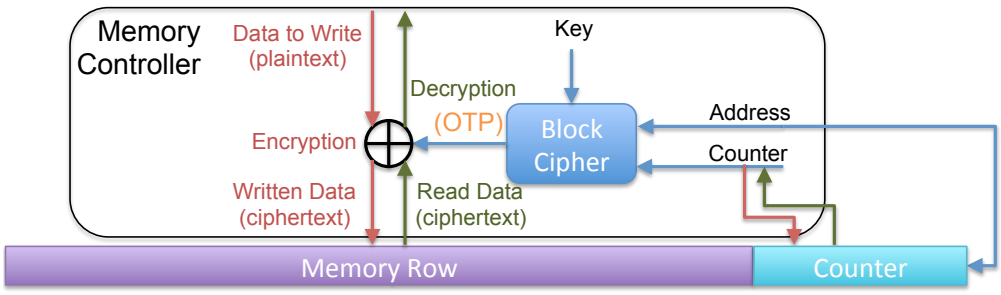


Figure 6: counter-mode encryption in the memory controller. Reproduced from [7], ©2020 IEEE

Memory systems which implement AES-GCM typically adopt an architecture as shown in Figure 6. The counter sits alongside the encrypted data in memory, which is manipulated and used with hardware-implemented AES to implement AES-GCM within the memory controller. With careful coordination of write queues, the AES block latency can be pipelined to take advantage of parallelism.

Unfortunately, the encryption process complicates its use with non-volatile memories. As noted in Section 2.3, encryption removes any patterns in the data, making it difficult to reduce writes by encoding or differential write. Likewise, encodings which target reducing energy-intensive symbols are defeated by the apparent randomness of the ciphertext being written back to memory. Every bit in memory has an equal likelihood of being '0' or '1', as does the new data being written, so on average we expect half of the bits to change in any given write.

DEUCE, or Dual Counter Encryption [37], attempts to overcome this by introducing the concept of an epoch, such that at the beginning of the epoch, the encrypted row is written directly, but thereafter only dirty words are encrypted and rewritten with a subcounter. SECRET or Smartly EnCRypted Energy EfficienT non-volatile memories [38] improves upon DEUCE by allocating a dedicated "sub-counter" for data blocks within the row. The sub-counter maintains independent count values for each block within an epoch. When a sub-counter saturates, the epoch ends—the main counter is advanced, all sub-counters are reset,

the entire row is encrypted, and the result written with the new count value. SECRET addresses reliability by allocating ECP per row to tolerate faults. DEUCE does not consider reliability and SECRET provides significant savings over DEUCE at the cost of additional sub-counter bits [38].

Another work proposes using the counter value to track how often cells have been written. As the counter approaches the rated endurance of the PCM cell, more capable error correction schemes are activated to prolong the useful life of the memory [39]. CASTLE & TOWERs is complementary to this approach as it reduces wear throughout the memory lifetime, prolonging the interval of time in which the lower-overhead ECC is used or helping to achieve a particular uncorrectable bit-error rate (UBER) with a lower overall ECC storage. Another approach is to co-design the encryption scheme and the wear-leveling scheme of the memory controller. A smaller counter can be used in tandem with the physical address if that address will only be written some fixed amount of times before remapping, resulting in improved overhead and latency [40]. Again, CASTLE & TOWERs is complementary since its reliability and energy advantages can be realized within a small window of writes to any given address.

These techniques collectively improve the performance of encrypted main memory systems, but other approaches have been standardized for non-volatile and hybrid memory systems to improve security at rest. In the next section, we describe the AES-XTS algorithm which was originally designed for hard disk encryption, and later adapted for use in commercial PCM [6].

2.4.2 The AES-XTS Encryption Algorithm

AES-CBC provides data confidentiality, but can be manipulated—if it is possible for a malicious actor to know parts of the plaintext data, say shared information on a system which always uses encryption, then the attacker can inject their own plaintext into alternating blocks of memory using a Malleability Attack [41]. For storage-class memories, the strategy adopted as the standard to mitigate such attacks is XOR-Encrypt-XOR Tweakable block cipher with ciphertext Stealing (XTS) [42]. This standard is combined with an existing block

cipher encryption algorithm (*i.e.,* AES) to ensure that data cannot be modified undetected in secure systems. This is especially important for storage-class memories, where some data accesses may be few and far between, and both confidentiality and authenticity of data must be guaranteed.

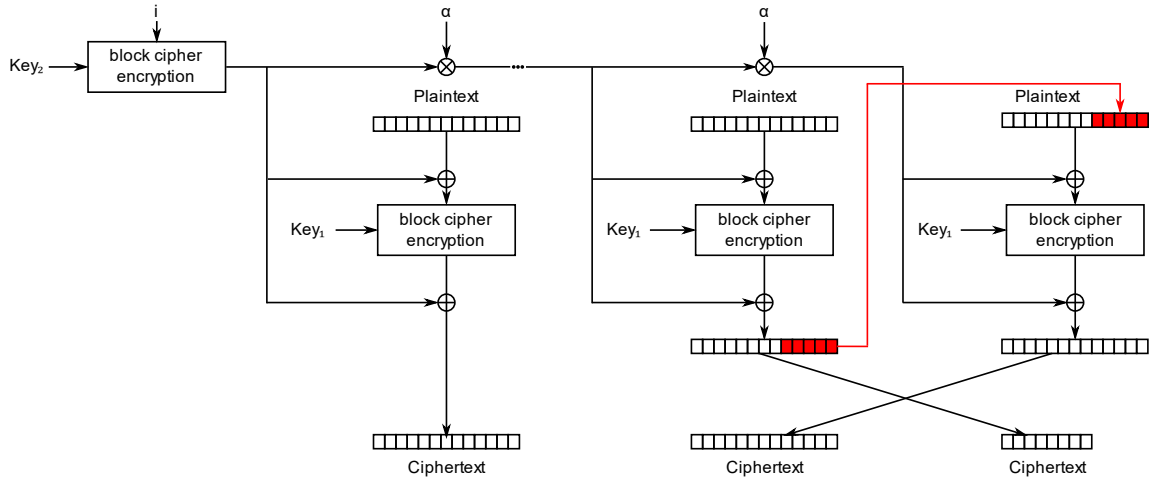


Figure 7: Example of XTS for block cipher encryption.

The key mechanisms of AES-XTS are an `XOR` operation before and after encryption—the XEX portion—and the use of part of the ciphertext in the last block's input—the ciphtertext stealing. Figure 7 depicts the basic architecture of XTS systems. Two keys are used for separate block cipher encryptions: one to generate a OTP from an initial value, fed forward to all encryption blocks (KEY_2 in Figure 7), and a second for use in encryption between the `XOR` operations (KEY_1 in Figure 7). Usually this is implemented with a single symmetric key (`e.g.,` 256-bit) split in half. The initial value is an arbitrary non-negative integer, but is essentially a constant function of the address being encrypted.

The first iteration of XEX was based on AES-GCM to exploit its performance benefits. However, vulnerabilities in this approach [43] necessitated the addition of a "tweaking" adjustment represented by the chain of multiplications by α in Figure 7. To support arbitrary data block sizes, the ciphertext-stealing approach was added, which pads the final encryption block input by a portion of the penultimate encryption block output, as shown in red. The resultant algorithm was adopted as a standard for disk encryption where data access control

or authentication via data expansion methods is not feasible [44].

Unfortunately, the additional transformations of the XEX and tweaking steps combined with the chaining of the final two blocks of the ciphertext-stealing step complicate the efficient application of cosets. MACE and its supporting architecture WINDU were designed to overcome these complications, providing an adjustable balance among write reduction, energy reduction, and performance.

In the next chapter, I present the concepts of CASTLE counter-mode encryption, my contribution TOWERs to efficiently encode encrypted data, and how they are used together to reduce cell writes, mitigate endurance faults, and reduce energy in encrypted memories using PCM technology.

3.0 CASTLE & TOWERs

"Counter Advance for Secure, Tailored Lifetime Extension" (CASTLE) is a technique which uses artifacts of the encryption process to extend the lifetime of encrypted PCM. CASTLE and its primary mechanism counter advance are based on AES counter-mode encryption: Each memory row has a counter value used to generate a OTP, which is subdivided into sub-counters associated with blocks of memory within the row. By incrementing sub-counters independently, a set of OTPs can be generated on a per-block basis. Using these blocks and the coset transformation techniques described in Section 2.3, CASTLE can mask stuck-at faults and prolong the lifetime of the memory.

"Techniques for Optimizing Wear and Energy Reduction" (TOWERs) is a novel encoding technique which leverages random cosets and compression to improve fault tolerance and reduce dynamic energy. Since counter-mode encryption preserves the plaintext until the final step of `XOR` by the OTP, TOWERs can apply lightweight compression to create space for encoding auxiliary information required by cosets. This allows TOWERs to efficiently evaluate several candidate words for writeback, which are then optimized for write reduction, fault tolerance, and energy reduction. Together with CASTLE, TOWERs can dramatically improve memory lifetime while opportunistically reducing energy.

Portions of this chapter reproduce text and images with permission from my publication "A CASTLE with TOWERs for Reliable, Secure PCM" [7] ©2020 IEEE. The concept "counter advance" and the derived technique CASTLE are presented herein for reference, but these are attributed to my co-author Donald Kline Jr. as an extension of a previous publication [45].

3.1 CASTLE Counter-mode Encryption

CASTLE's primary mechanism *counter advance* leverages the property that encryption of plaintext with a new counter generates a new random ciphertext. Recall that in the

event of a PCM endurance failure, data which matches the stuck-at value is SA-R and data opposed to the stuck-at value is SA-W. When writing a ciphertext candidate in the presence of stuck-at faults, each faulty cell has a 50% probability of being SA-R, and similar for SA-W. Thus, by incrementing the counter, it is possible to improve fault tolerance by finding a ciphertext candidate that maximizes SA-Rs.

Consider the example in Figure 8 for a row with two stuck-at faults such that, given the ciphertext, the first is SA-R and the second is SA-W. Advancing the counter (`Counter+1`) resulted in the first fault becoming SA-W and the second becoming SA-R. This is due to the property that each fault in each ciphertext candidate has a 50% chance to be SA-R, but is equally likely to be SA-W. Advancing the counter again (`Counter+2`) was unlucky, resulting in two SA-Ws. The probability of finding an error-free candidate with f faults is 2^{-f}, or 25% for f=2, which required multiple advancements in the example. Both word-level encryption and error correction can dramatically reduce the number of advancements to find an error-free candidate. For example, with single bit error correction (e.g., ECP-1), the example of Figure 8 would have been successful without counter advancement. If SECRET is used with independent sub-counters per block and assuming all blocks were dirty, blocks zero to two would have been written with `Counter` and block three would have used `Counter+1`. Since a sub-counter exists for each block, only the sub-counter for block three would be advanced.

The application of CASTLE with block level encryption and error correction is shown in Figure 9. Figure 9a expands on Figure 6 with sub-counters per 128-bit block in the style

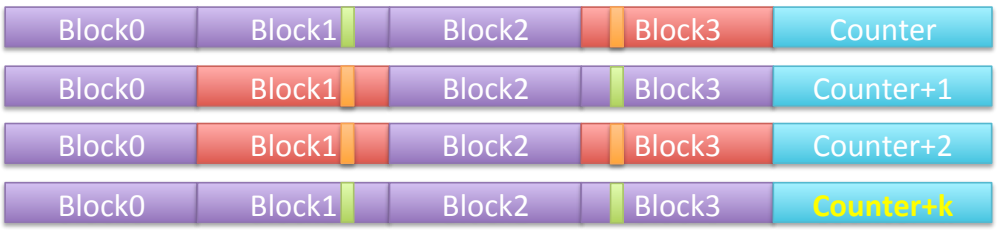
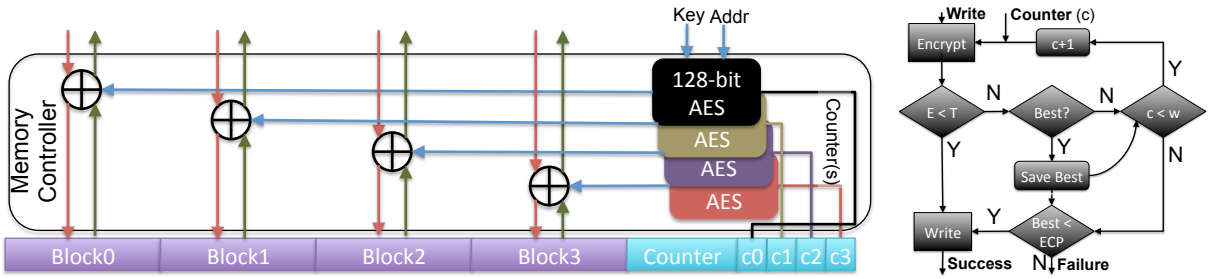


Figure 8: CASTLE example for one memory line. Green indicates a SA-R fault and orange a SA-W fault. Purple blocks are error free and red blocks contain an error.



(a) Encrypting individual blocks with sub-counters.

(b) ECP encoding flow.

Figure 9: Block level CASTLE architecture.

of SECRET [38]. CASTLE applied in this context examines the stuck-at bits independently between blocks and only advances the counter when SA-W bits appear. Stuck-at faults can be determined by storing and reading patterns of all '1's and '0's or using a fault cache [31] or fault map [27]. It is straightforward to extend block-level encryption with ECC as the parity bits (e.g., SECDED [64,72] ECC) would not cross block boundaries. In this case, CASTLE could protect fewer SA-W errors and allow ECC protection to correct others at the cost of reduced transient error protection.

ECP, in contrast, uses pointers that are shared by the blocks of a given row. Block-level CASTLE offers the trade off of using a pointer or advancing the counter to mitigate a fault. Using a pointer will reduce the availability of pointers to tolerate faults in other blocks. The algorithm for selecting the appropriate write candidate for ECP with CASTLE is shown in Figure 9b. Since auxiliary bits for ECC parity or ECP are designated per-row and must be stored in plaintext form, CASTLE does not interfere with the operation of either error mitigation technique.

Assuming a counter advancement epoch window w where $w = 2^b$ and b is the number of bits for each sub-counter, w serves as a threshold of how many counter values will be explored to accomplish a particular write successfully. If the encrypted data experiences E errors (*i.e.,* SA-Ws) but E is less than a threshold T, the write proceeds with the current counter value c. Otherwise, if this is the "best" candidate seen so far (*i.e.,* fewest SA-Ws), it is retained. If c is still within the epoch w, c is incremented and the next candidate is

evaluated. If c reaches the limit of the epoch without finding a candidate within the error threshold, the best candidate is written if sufficient ECP pointers are available, otherwise the write fails. In our evaluation we consider two schemes: the *counter minimization* (CM) approach sets T to the number of available ECP pointers, allowing a write to proceed with the minimum counter value that discovers a possible solution, while the *pointer minimization* (PM) approach sets T=1 requiring a fault free solution to write immediately.

In the next section, I present the concept of TOWERs, how it leverages compression to reduce overhead associated with coset encoding, and how it is used to improve lifetime and dynamic energy in encrypted PCM.

3.2 TOWERs

Due to the encryption process, the output written back to memory has been effectively randomized; any existing similarities in the plaintext data will be obscured and distributed among the bits of the ciphertext by the encryption method. CASTLE leverages this property to generate new storage candidates that better match the faulty PCM cells. However, this randomness presents two challenges. The first challenge is that these randomized storage values defeat techniques which rely on similarity. The second challenge is that the randomness incurs more bit changes, in turn increasing write energy and accelerating wear.

One strategy to minimize the resulting bits written to preserve endurance (and minimize dynamic energy) is to use cosets to encode the data before writing it back, as presented in Section 2.3. This applies for plaintext data that has similarity [46], apparently random data [47], and hybrid data that contains both [34]. This strategy can similarly be an effective way to extend the lifetime of a memory in the face of stuck-at bits. In the context of encrypted data, the use of a biased code which performs well for plaintext will not benefit, and may even hamper encoding goals. For strictly unbiased random data, it is more effective to use a random code [34].

Unfortunately, employing cosets of any kind introduces additional overhead bits which are necessary to encode the specific coset candidate applied to each word. These bits consume

the available capacity for users or the error correction scheme, and writing these bits may degrade the energy benefit. One method to mitigate this overhead is to leverage lightweight compression of the data and make room for storing these auxiliary bits in the cache line itself.

3.2.1 Compression & Encoding of Encrypted Data

Several such compression schemes have been proposed, which typically leverage common patterns of the data for their compression [46, 48]. The challenge for encrypted data is that it will not exhibit these patterns required for effective compression. Instead, data can be compressed prior to encryption. The coset can then be applied to the compressed data and the reclaimed space can be used to store the encoding. Using block-cipher encryption, each coset candidate must then be passed through the cipher to create a candidate for writing. For N cosets applied over M blocks, $(N-1) \cdot M$ additional encryptions would be required to generate and select the best coset candidate, which is not scalable.

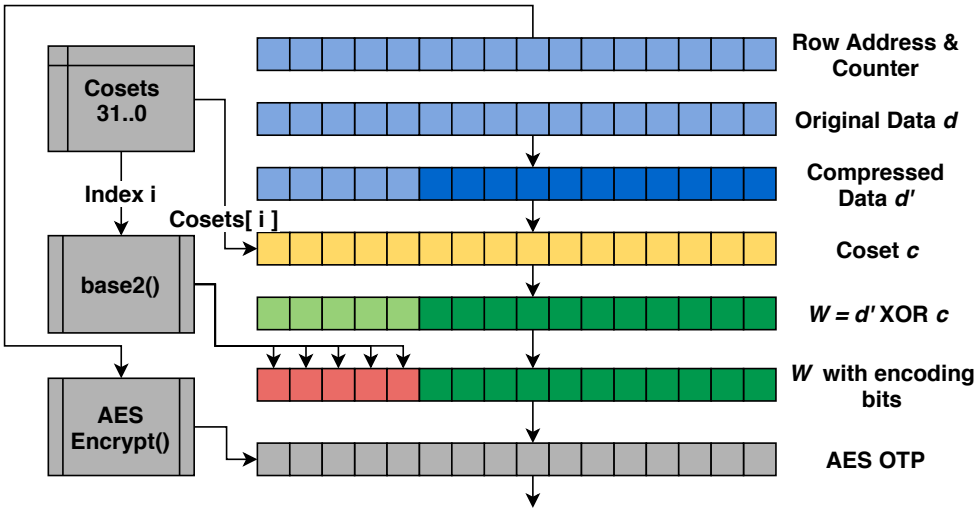


Figure 10: Generating a writeback candidate for TOWER using the counter-mode AES output, the compressible plaintext data, and a table of coset codes.

In counter-mode encryption, as shown in Figure 6, the AES block cipher block is generated once per cache line and applied via the OTP. Thus, each encoded coset candidate can

be encrypted with the cost of a single XOR operation, allowing us to utilize the compression and encryption in the same process. Figure 10 demonstrates this process: First, the data d is compressed to to yield the compressed data d' in dark blue and the reclaimed bits shown in light blue. Next, in parallel, each coset candidate (yellow) is applied to a compressed word via XOR (green), and appended with the binary coset ID (lightgreen→red). Finally, the AES OTP is applied, also via XOR (gray), and the resulting data is evaluated for writeback.

The effectiveness of TOWERs relies on compressibility of the words within the cache line. In contrast with general compression needs, here compression coverage is more important than compression ratio. Compression coverage is the percentage of cache line writes that can be encoded in compressed form, which is distinct from compression ratio which describes the ratio of data length before and after compression. To achieve a higher coverage, I implemented a relaxed version of MTC [48] which only attempted to recover five bits per word for all compression cases. Standard MTC attempts to discover if the most significant eight bits of each word in a cache line are either (1) uniform (all 1's or 0's) or (2) identical for each word in the line. Because of the needs of our encoding, we relax this to only requiring the leading six bits to be uniform or identical across all words of the line. This scheme requires two auxiliary bits per cache line to signify that the cache line was compressible and to encode which compression scheme was applied.

Given a particular coset, a encrypted coset candidate can be selected to optimize one of many factors such as energy [47, 46] and endurance, among others. Given that the primary concern for PCM is to maximize fault tolerance, the most appropriate candidate for writeback is one where the count of SA-W bits is minimized, to maximize the likelihood that the remaining faults can be addressed by the error correction scheme.

However, given the limited endurance of the memory, during the memory lifetime, the fault rate will change and the number of stuck-at bits encountered will increase. This will reduce the probability of any given coset to correct the SA-W bits, and in turn reduce the number of candidates available. Thus, during the portion of the lifetime where there are multiple acceptable fault tolerance candidates, there is an opportunity to employ coset techniques to improve endurance and potentially also reduce energy. Rather than simply choose the first acceptable candidate, we propose selecting the best candidate more intelligently.

Just as differential write extends the PCM lifetime by reducing the stress on endurance [49], selecting the best coset candidate that mitigates faults and optimizes endurance can similarly improve lifetime. Furthermore, since RESETs require more energy than SETs [17], and are more detrimental to the endurance of PCM, we benefit from a reduction of RESET operations.

If multiple candidates are available that either minimize SA-Ws, or that have SA-Ws that can all be addressed by error correction, a candidate can be selected that also optimizes other metrics. This multi-objective coset optimization approach is discussed in the next section.

3.2.2 Multi-objective Coset Encoding

Among the acceptable candidates from a fault tolerance perspective, we can define the best candidate in terms of a secondary optimization parameter. In this context, the count of RESETS incurred (or the differential write energy required) could be used to rank coset candidates with the same fault tolerance. At any given time, there may be multiple solutions which meet the SA-W constraint. However, determining the optimal coset candidate is cost-prohibitive. Instead, we adopt a heuristic to pare down the candidates based on SA-W minimization. In the case of a tie, the coset candidate that also minimizes the secondary metric, in this case endurance—modeled as the number of RESET bits—can be selected.

We can improve on this approach if we relax the reliability constraint. If the number of SA-W bits that can be corrected by the memory controller's correction scheme is higher than the number of faults encountered, it may be reasonable to accept a higher number of faults if we have an overall improvement in the secondary metric. For example, if there were five SA-W faults in a cache line, for a memory controller using ECP-6, then the pool of acceptable writeback candidates could be extended to include those with six SA-W bits, but only if the impact of utilizing those extra encoding bits results in a net decrease in RESETs.

Algorithm 1 conceptually defines the heuristic. For each word we track two fault classes, the minimum and second minimum number of faults across all the coset candidates. Within each class, the best coset is selected such that the overall cost of the secondary parameter

```
Data: cosets[N], best coset b, next best coset nb
1  for c ∈ cosets do
2      if countSAW(c) < countSAW(b) then
3          nb ⟵ b
4          b ⟵ c
5      else if countSAW(c) == countSAW(b) then
6          if cost(c) < cost(b) then
7              b ⟵ c
8          end
9      else if countSAW(c) < countSAW(nb) then
10         nb ⟵ c
11     else if countSAW(c) == countSAW(nb) then
12         if cost(c) < cost(nb) then
13             nb ⟵ c
14         end
15 end
```

Algorithm 1: Selecting a coset candidate for each word

is minimized. Both a best and next best coset candidate are tracked at the word level. In particular, lines 2-4 check to see if the current evaluated candidate has better SA-W than the current best candidate, and if so keeps this as the new best candidate and demotes the previous best candidate to next best. Lines 5-8 check and keep the current candidate if it has the same SA-W bits and a better secondary cost function. Lines 9-10 keep the candidate as the next best candidate if it has less SA-W than the previous next best candidate. Lines 11-13 keep the candidate as the next best candidate if it has the same SA-W but a better secondary cost as the previous next best candidate. The algorithm can be applied in parallel over all words in a cache line, and the testing procedure can use a tree-wise reduction to obtain the top two coset candidates for each word.

For the secondary cost metric to overcome the increase in SA-Ws, the benefit of selecting a coset candidate with more SA-W bits should outweigh the cost of using the additional correction bits. Consider the case of RESET bits for minimizing endurance. For a 512-bit cache line, ECP requires nine bits of address plus one bit as a spare. We assume that the address bits and spare bit have an equal likelihood of being '1' or '0' for any given fault. Thus, using an ECP entry will require writing half of those bits on average. Likewise, there is a 50% chance that the replacement bit already matches the value it is to store. Of the 50% of correction bits changing, on average half will be SETs and half RESETs. So for a cache line of n bits, we can define a threshold T which captures the number of RESETs that must be reduced by the coset as:

$$T = \left\lceil \frac{1 + \log_2(n)}{4} \right\rceil \cdot RESETs \tag{2}$$

If average energy was to be the secondary metric we would replace $RESETs$ in Eq. 2 with $(E_{SET} + E_{RESET})$ where E_{SET} and E_{RESET} are the energy cost in joules for a SET and RESET operation, respectively. If we can observe an endurance improvement (or energy reduction) greater than this threshold by some margin, we expect to see an improvement by switching to a coset code which results in the additional SA-W bit(s).

There is the potential to end up with more SA-W bits per cache line than the error correction scheme can correct with Algorithm 1. In a reasonable worst case, we expect one additional SA-W per word, which means eight additional SA-W bits per 512-bit cache line. To avoid this situation, if the selected next best candidates would cause the write to fail (more total SA-W than the correction scheme can correct), then only the best candidates are chosen. This will allow the algorithm to opportunistically reduce the secondary cost, while avoiding taking on more SA-W bits than the correction scheme can support. In this way, coset techniques can be applied to encrypted data to mitigate stuck-at faults while reducing wear and energy consumption.

3.3 Experimental Setup for CASTLE & TOWERs

To evaluate CASTLE and TOWERs we created a modified Pintool [50] to monitor all writebacks encountered at the lowest-level, unified cache. Using the instrumentation, Pin was attached to runs of individual SPEC2017 benchmarks [51], extracting the address and data of 512-bit cache lines as they were evicted. The aforementioned benchmarks were selected to be representative of general contemporary and future workloads. Among the SPEC2017 benchmark suite, the benchmarks in Table 1 provide excellent coverage of the full suite by virtue of their similarity to the remaining benchmarks[52]. Also included were any benchmarks which exhibited a relatively high percentage of load and store operations. All benchmarks were compiled using the SPEC base and speed metrics, and executed using reference workloads. SPECCPU 2017 workloads were selected over 2018 IaaS Cloud benchmarks for this experiment as the former was found to be more appropriate for testing and evaluating a single server instance [53].

Table 1: SPEC2017 benchmarks used to generate memory traces

Benchmark	**Application Domain**
602.gcc_s, 600.perlbench_s	Compiler
625.x264_s,657.xz_s	Compression
605.mcf_s	Combinatorial Optimization
620.omnetpp_s	DE Simulation
623.xalancbmk_s	Document Processing
603.bwaves_s, 619.lbm_s	Fluid Dynamics
621.wrf_s	Climatology
631.deepsjeng_s, 641.leela_s	AI

When testing the reliability of proposed and existing correction schemes at fixed fault rates, we used tools from PREMSim [54] to generate fault *snapshots*, which matched five predefined fault incidence rates: 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}. These fault snapshots represent different points along the lifetime of the memory. They describe faulty bits stuck

at '0' or '1,' and use a Bayesian distribution to mimic the impact of process variation with spatial correlation of faults [25]. Each fault snapshot was prepared for a 2GB memory, where fault locations are specified at the bit level (row i, column j). Under simulation, fault vectors were created from the fault snapshot for each cache line writeback based on the trace address.

To evaluate the effectiveness of CASTLE and the counter advance concept, SECRET was used as the comparison baseline [38] assuming three bits were allocated as a sub-counter for each block, setting $w = 8$. Note, SECRET assumes ECP-6, which we assume for the baseline unless otherwise stated. Snapshots of high cell failure rates (10^{-3} and 10^{-2}) representing later points during the memory lifetime were used as stimuli for counter advance. A "word-level" 64-bit block size was selected to match the word size in modern architectures.

As noted in Section 3.4.3, the effectiveness of TOWERs relies on compressibility of the words within the cache line. To determine the compression coverage of a given cache line writeback, we applied our relaxed version of MTC [48] to compress writebacks generated from each of the workloads to completion. We also determined smaller 1M and 5M access windows during each workload that best matched the overall compressibility of the workload. These sample windows were used for TOWER experiments that evaluated the impact of coset encoding. When using TOWER with a compressible cache line, we have the potential to utilize different numbers of coset codes. Depending on the compression ratio, the reclaimed bits m can encode 2^m codes. A larger number of coset codes provides more opportunities for fault correction, but we expect diminishing returns beyond a certain number of codes. To determine the appropriate number of codes to use, memory traces were evaluated using sets where $m \in 2..7$, *i.e.,* sets of 4 to 128 code words. Randomly generated cosets were used to generate writeback candidates across 1M representative cache line writes, and the mean across benchmarks was used to assess tradespace between coset size and SA-W reduction.

The cosets in TOWER are not only beneficial for reliability. As described in Section 3.2.2, memory traces were evaluated via Algorithm 1 to optimize SA-W bits as a primary optimization objective, and then either differential write energy or RESETs as a secondary optimization objective. To assess which of these secondary objectives was most beneficial, we observed SA-W bits, total write energy, and the total count of RESETs, over 5M cache

line writebacks. A set of 31 pseudo-randomly generated coset codes were fixed for the duration of the experiment. An identity code of all '0's was used to allow the cache lines to be written back relatively unperturbed. For each writeback, the control case applies baseline SECRET, which uses counter-mode encryption to the plaintext data, then evaluates the above criteria on the dirty words requiring writeback. The experimental case first attempts to compress the plaintext input via our relaxed MTC [48]. If the line is not compressible, the writeback is processed as with the control case. Otherwise, a coset is selected via Algorithm 1, the best coset is applied, the counter-mode encryption is applied, the coset ID is encoded in the reclaimed bits, and finally the above criteria are evaluated. In both cases, a randomly-initialized memory tracks the last data written back to each address, such that the differential write energy is accurate. Note, if MTC changes compression methods, all words are considered dirty and must be written.

In order to understand the impact of PCM feature size on write energy, we tracked the number of SET and RESET operations incurred for each of the workloads both with and without CASTLE with TOWERs. Using the reported [55] and derived [17] write energy parameters for existing and future feature size, this permits a comparison of how the scaling of write energy influences the effectiveness of the secondary optimization parameter in TOWERs.

To estimate the overhead of the coset approach, Verilog implementations and ROM look-up tables were synthesized using Synopsys Design Compiler targeting 22nm technology and CACTI [56] was used to model and estimate SRAM storage, following a similar methodology in the literature [57].

To experimentally evaluate the effect of our techniques on PCM lifetimes, our lifetime study examined improvements compared to SECRET and encrypted data protected with AEGIS. We assumed that the distribution of cell lifetime in PCM followed a normal distribution with spatial correlation of faults [24, 25], with a mean failure rate of 10^8 writes [32, 58]. We assumed a base coefficient of variation of 0.2 [32, 58]. We also conducted a sensitivity study with higher variations of 0.25 and 0.3 to approximate the effect of scaling on process variation. As process variation increases with scaling, the memory will incur cell faults more quickly and better error correction will be necessary to maintain effective lifetimes. The co-

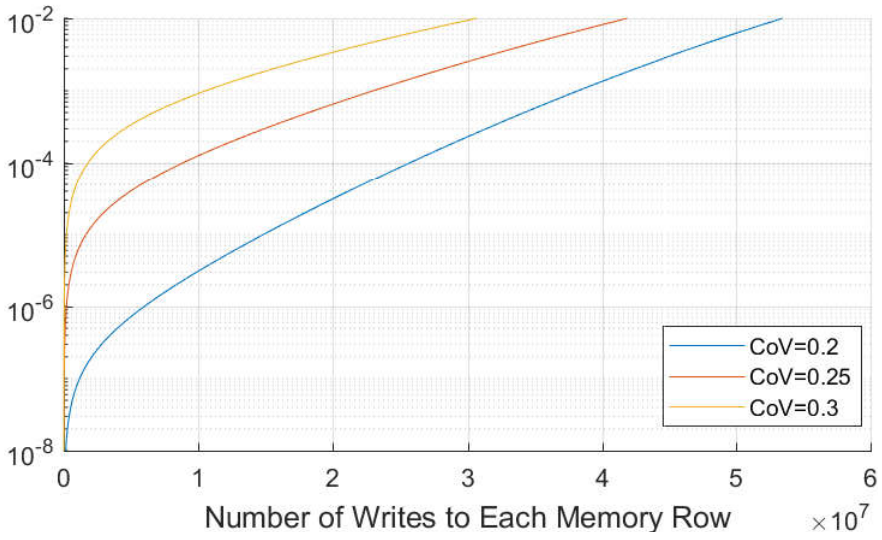


Figure 11: Cell fault rate for different coefficients of variation.

efficient of variation of cell failure distribution is a commonly used representation of process variation, as shown in Figure 11, where with increasing process variation 10^{-2} fault rates will be approached with fewer writes. The lifetime study occurred on a 1MB segment of the memory with 4kB pages, consistent with prior work [58] with memory row sizes of 512 bits. We assumed a perfectly uniform wear-leveling approach where each address receives an equal amount of writes, which serves as an approximation for proposals in the literature which approach [59, 60] or exceed [58] this standard. Wear leveling is combined with differential write to reduce the frequency of cells written for each operation. Under these conditions, the simulations continuously issue writes to memories protected by the techniques under study, until there are four rows with unrecoverable faults [59], which delineates the memory lifetime such that we can determine their lifetime improvement. The results shown are an average of 10 lifetime experiments.

Finally, to model the performance impact of CASTLE and TOWERs compared to the SECRET baseline, we used the SNIPER full-system simulator [62] with the parameters shown in Table 2. To understand the impacts of PCM device parameters on performance and energy we reported results for a device at 90nm [55]. We also conducted sensitivity studies for

Table 2: Architecture parameters for performance study.

CPU	**Cache**
4 out-of-order cores	Private L1 32kB inst., 32kB data
4 issue width, 4GHz clock	Private L2 256kB/core
28nm technology	Associativity: 8 (L1 data and L2)
1GHz frequency	Block size: 64B
Memory: PCM	
512-bit row, 64-bit words, 2GB main memory	
2 channels, 1 rank/channel, 8 banks/rank	
4 AES units, 111ns delay [61]	
TOWER+32 cosets encode/decode: 2.63ns/0.89ns [46]	

energy savings and performance impact for 180nm, 90nm, 45nm, 32nm, and 28nm using PCM devices reported in the literature [55, 17] and modeled using peripheral circuitry reported by NVSim [63].To model a system employing CASTLE and TOWER, we used estimates of encryption delay from [61] and estimates of coset delay from the hardware implementation study, assuming up to four AES operations could be performed simultaneously.

3.4 Results

Based on the experimental setup described in Section 3.3, we conducted detailed evaluations of CASTLE and TOWERs individually to see the impact of each contribution. We then combined them together to evaluate a full system benefit.

3.4.1 CASTLE Study

Figure 12 shows a summary of the uncorrectable bit error rate, defined as the number of bit errors that occur per bit written, for both row (CASTLEr) and block-level (CASTLEw) encryption with different strengths of error correction used to protect the data and counter bits. With no error correction, word-level CASTLE (CASTLEw) provides two orders of magnitude improvement in UBER compared to word-level encryption alone for as high as a 10^{-3} cell failure rate. As error correction is employed the improvement is amplified, providing 3–5 orders of magnitude improvement by introducing one ECP pointer (ECP-1). At a cell failure rate of 10^{-3}, an UBER of $\leq 10^{-11}$ required only ECP-4 for word and row CA. Employing PM reduced the requirement to ECP-3. In contrast, SECRET, which includes ECP-6, can only achieve a $4 \cdot 10^{-10}$ UBER.

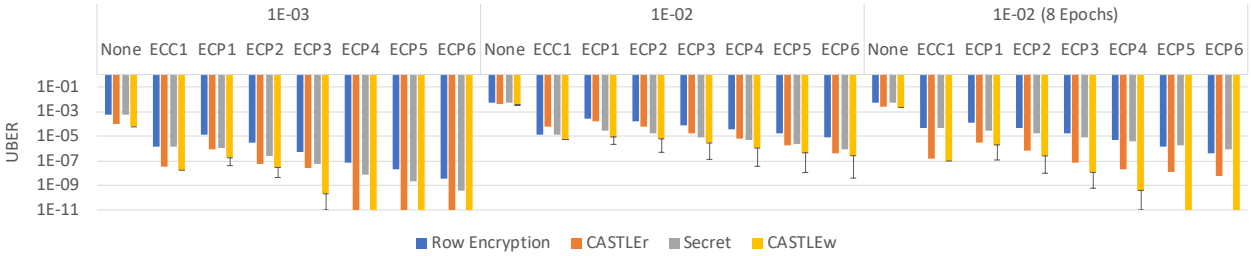


Figure 12: UBER for various error rates. CASTLE explores one epoch (w=8), except where noted. CASTLEw+ECP is reported for CM with an error bar indicating PM.

At a cell failure rate of 10^{-2}, unsurprisingly, UBER is drastically reduced. Using ECP-6, CASTLE achieves almost a 10^{-9} UBER versus 10^{-6} for SECRET. Relaxing CASTLE to explore eight epochs allowed CASTLEw with PM to function at a $<10^{-11}$ UBER with ECP-4 and higher. This indicates that while a device might operate using the CM approach initially to minimize counter advancements, it could switch into PM and expand the searching window for *gracefully degraded* operation when the cell failure rate became sufficiently high.

CASTLE is sensitive to block size. Small block sizes increase the flexibility to eliminate faults. A sensitivity study indicates CASTLE is nearly as effective for 64-bit blocks as 32-bit blocks. 128-bit blocks have a noticeable degradation (0.5-1 orders of magnitude UBER), particularly with ECC and ECP, however, the CASTLE improvements are still dramatic.

A logical concern about CASTLE is the impact to performance from evaluating multiple ciphertext candidates and the potential to saturate the encryption counter more quickly. Figure 13 shows the number of counter increments per write operation. Word-level encryption naturally reduces the average counter advancements per write (A) to A=0.96 compared to the row-level baseline of A=1, as each write only advances the dirty words' sub-counters. This provides sufficient "room" for word-level fault-induced counter advancements for lower fault rates (*e.g.*, $\leq 10^{-4}$) without exceeding the row-level counter lifetime.

At 10^{-3}, for ECP $\geq$3, A<1. At 10^{-2} there are significant fault-induced counter advancements, owing to gracefully degraded operation. However, increasing the number of epochs searched for larger numbers of ECP pointers, especially ECP-6, provides significant improvements in protection, with only slight increases to A. To achieve an UBER of 10^{-11} with ECP-3 only requires A=1.07 after a cell failure rate of 10^{-2}.

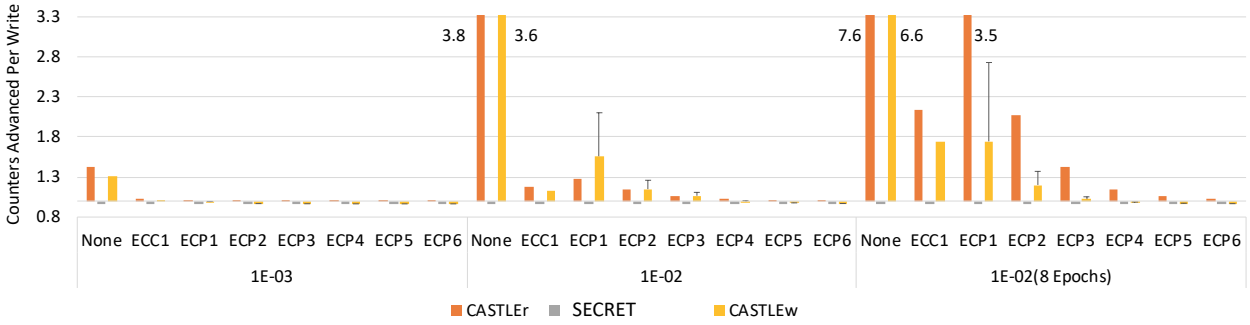


Figure 13: Counters advanced per write at various error rates. CASTLE explores one epoch (w=8) except where noted. CASTLEw+ECP is reported for CM with an error bar indicating PM. Row Encryption is always unity.

As this gracefully degraded mode would only occur very late in the memory lifetime, these counter advancements would only saturate the counter nominally sooner and minimally increase encryption energy, while extending the usable life dramatically. If the system is reset with a new encryption key or the data is moved for another reason (e.g., wear-leveling [40]), the counter can also be reset. The performance impact is studied in detail in Section 3.4.6.

Counter-mode encryption has a downside that it requires the storage of a counter for each row. Unfortunately, this overhead cannot be eliminated for counter-mode encryption.

However, the storage dedicated to the per-word sub-counters, initially proposed by SECRET [38] to reduce energy and improve endurance, could be retargeted to improve fault tolerance. This storage would be insufficient to add additional ECC, but could add two additional ECP pointers. This comparison is shown in Figures. 12 and 13 by comparing CASTLEw with ECPN to CASTLEr with ECPN+2. The results indicate that for lower fault rates, CASTLEr would provide an advantage in fault tolerance at the cost of increased energy and reduced endurance. As the fault rate increases, CASTLEw in PM mode is more fault tolerant while maintaining energy and endurance benefits over CASTLEr.

3.4.2 Coset Size Study

Figure 14 depicts the mean count of SA-W bits as the number of coset codes is swept from 4 to 128 for a fault incidence rate of 10^{-2}. Similar results were obtained for 10^{-3} down to 10^{-6}. The plot represents the mean of 1M cache line writebacks across the SPEC2017 benchmarks. The curve exhibits diminishing returns for a count of cosets beyond 32, with the knee between 16 and 32 cosets. This aligns well with the amount of encoding bits recovered under MTC; all compression methods recover at least five bits per word. We conclude that a coset of 32 codes is ideal for applying the compression-optimized coset approach.

3.4.3 Compressibility Study

Figure 15 depicts the compressibility of the SPEC2017 benchmarks using MTC. Each benchmark was evaluated for the first $5 \cdot 10^9$ cache lines or until the end of the trace was reached. Several of the benchmarks exhibit high compression coverage, which is promising for the application of TOWERs. Under simulation, the number of cache lines was restricted to a representative sample of 5M cache line writes. Compressibility is relevant because the ratio of compressible lines is directly proportional to how often cosets can be applied. We thus expect to see a proportionally smaller improvement over the control case for workloads which are less compressible.

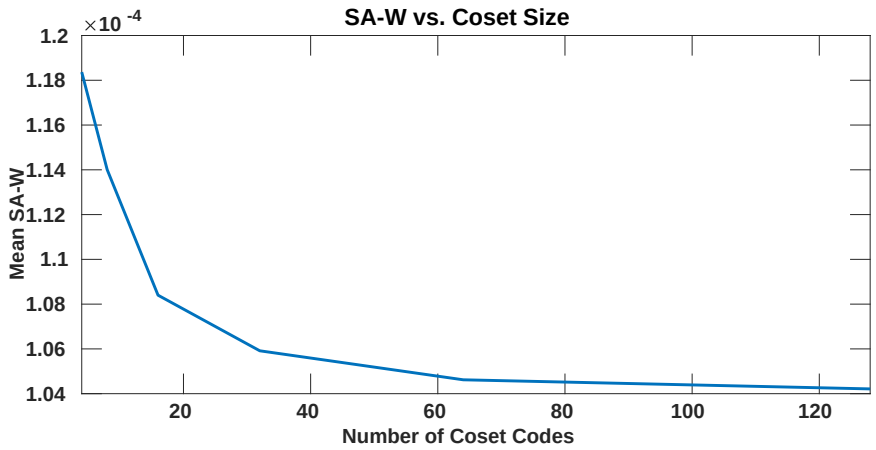


Figure 14: SA-W Bits vs. Number of Coset Codes, evaluated over 1M representative write-backs at a fault rate of 10^{-2}.

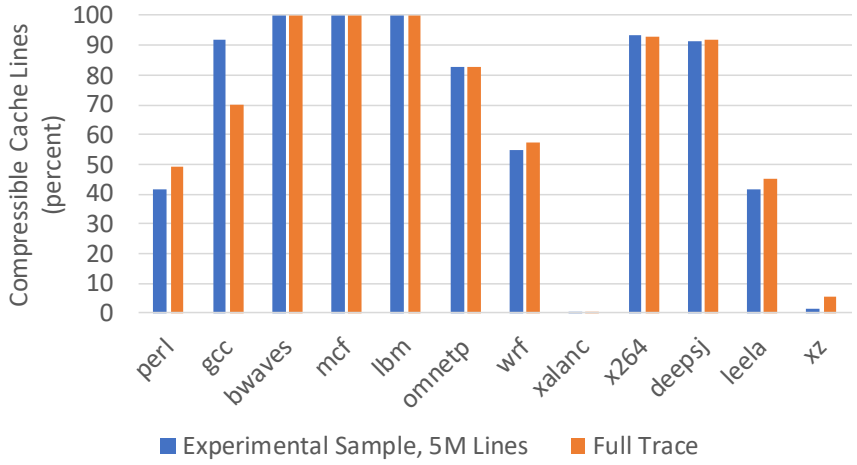


Figure 15: Compressibility of cache lines in the SPEC2017 traces, for the full trace and the sample under test.

3.4.4 TOWER Multi-Objective Optimization

To evaluate the effectiveness of TOWER, we consider first its impact on energy and endurance. Figure 16 depicts the change in differential write energy and of RESETs as

a percentage of the baseline of SECRET for each benchmark, using a fault rate of 10^{-4}. Each experiment first optimized for SA-W bits, and then optimized for either differential write energy or RESETs (as a proxy for endurance). Note SECRET achieves an average 46% reduction in write energy compared to row-level encryption, including encrypted data protected by partition-and-flip approaches such as AEGIS. As expected, we observe that the improvements follow the compressibility trends of Figure 15, as TOWER only operates on compressible rows. We also observe that optimizing for RESETS and for differential write energy produces comparable results for overall differential write energy, likely due to the higher energy of RESETs versus SETs [17]. We ran identical experiments for different fault rates (10^{-2} to 10^{-6}) and numbers of pointers (ECP-1 to ECP-6) and the trends are similar to the results from Figure 16.

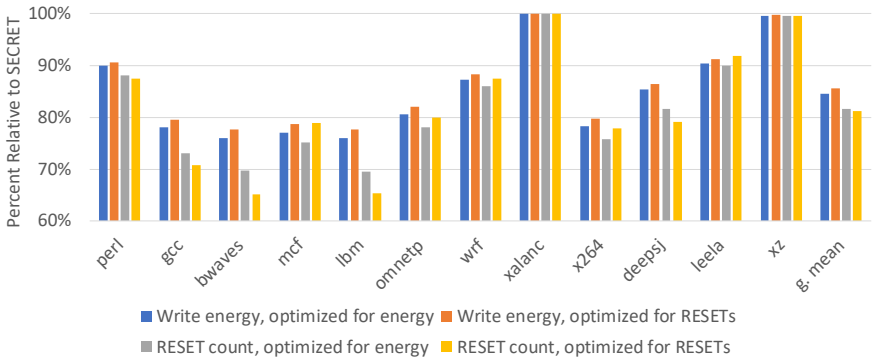


Figure 16: Change in RESETs (endurance) and write energy (lower is better) relative to SECRET at a fault rate of 10^{-4}.

Using reported and derived energy parameters for PCM at various feature sizes from 180nm down to 28nm [17, 55], we conducted a write energy sensitivity study on our optimized write energy as feature size scales[1]. Figure 17 reports CASTLE with TOWERs written energy reduction normalized to a control without optimization for a fault incidence rate of 10^{-4}, with RESET reduction as the secondary optimization parameter. Generally, the study indicates that the improvements from TOWERs remains highly correlated to the compression ratio and that the different devices does not dramatically change the trend. The older feature

[1]Three devices at 180nm, 90nm, and 32nm were modeled using reported parameters [17, 55]. Results for 45nm and 28nm were reported by scaling the 32nm device [17].

sizes 180nm and 90nm tend to show slightly lower benefit than the newer feature sizes from 45nm down to 28nm. This is due to a slightly higher ratio of SET energy to RESET energy at 180nm and 90nm (circa 0.7). This ratio improves for the smaller feature sizes (circa 0.5), resulting in a slightly better overall write energy reduction.

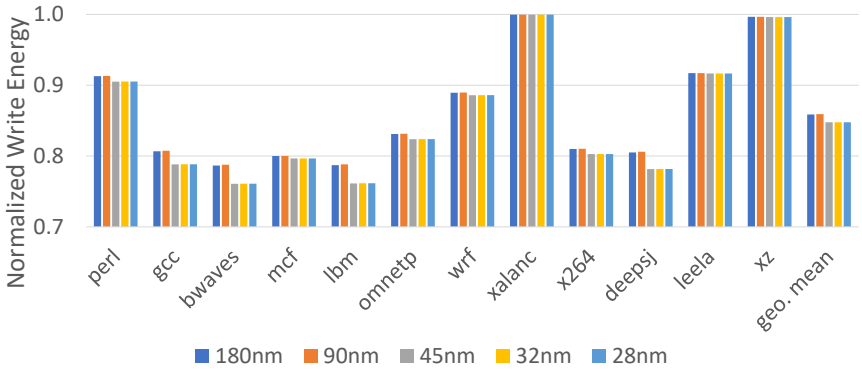


Figure 17: Write energy vs. feature size at a fault rate of 10^{-4}. Values are normalized to the control case (unprotected, encrypted writeback), with reduction of RESETs as the secondary optimization parameter.

We then considered the hardware required to realize these improvements. The auxiliary bits for multi-objective optimization compose an area overhead of 0.039mm^2, or <2.5% per GB in commercial PCM with a density of 0.64GB/mm^2 [5]. The coset encoding delay and power are 1.07ns and 15.8mW, while decoding is 0.35ns and 1.5mW. This is <1% of the write and read latency and power, respectively, for PCM reported by NVSim [63], <0.1% of the values reported for Optane DIMMs and is consistent with other results found in the literature [46, 57].

Of course, the primary goal of TOWER is to reduce the number of SA-W bits. To this end, Figure 18 demonstrates that the coset approach is effective at reducing SA-Ws when the data can be compressed. For a SECRET-like encrypted memory system using ECP-3, when applying TOWERs, highly compressible benchmarks like bwaves, mcf, and lbm achieve an improvement of several orders of magnitude for higher fault rates. There were comparatively fewer, if any, SA-W bits for the lower fault incidence rates and higher ECP protection, which is why ECP-3 is shown. When there were SA-W faults that ECP alone could not resolve,

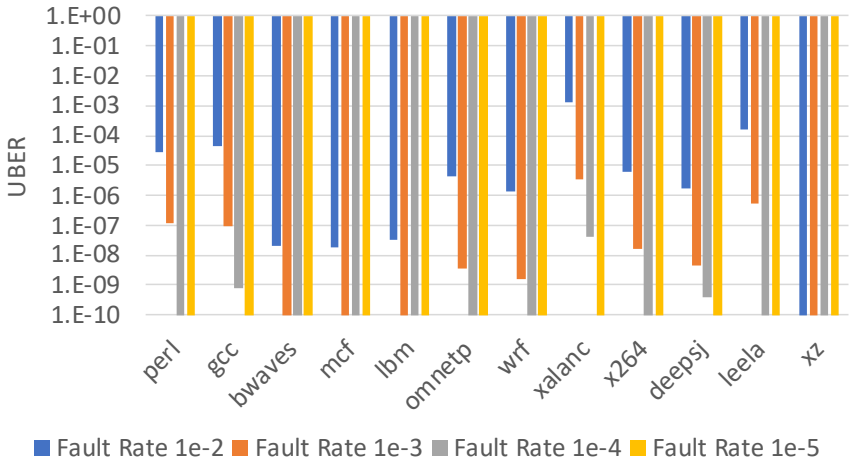


Figure 18: UBER using TOWERs and ECP-3, using RESETs as a secondary optimization parameter.

TOWERs effectively masked these faults and in turn reduced the UBER. Moreover, for fault rates lower than 10^{-5} all potential faults were eliminated.

3.4.5 Lifetime Study

Figure 19 shows the relative lifetime improvement of various correction schemes compared to ECC-1 at approximately the same overheads of ECC-1 (and ECP-6). CASTLEr refers to row-based CASTLE, and CASTLEw refers to word-based CASTLE. On average, ECP-6's benefits over unprotected and ECC1 are 70× and 4.4×. With the inclusion of SECRET, which includes ECP-6 but reduces the RESETs per write, this benefit over ECC1 increases to 6.4×. In this evaluation we also included an iso-area comparison with AEGIS, which can correct a guaranteed 11 faults compared to six for ECP-6. As a result, AEGIS provides a 12.4× improvement over ECC1 and a 1.9× improvement over SECRET.

The addition of CASTLEw achieves over 30× lifetime compared to ECC1, a 4.7× improvement over SECRET and 2.4× improvement over AEGIS. Further, the lifetime results highlight the benefits of TOWER: in the best cases with very high compression, adding TOWER to CASTLEw can achieve over 900× and 80× the lifetime compared to an un-

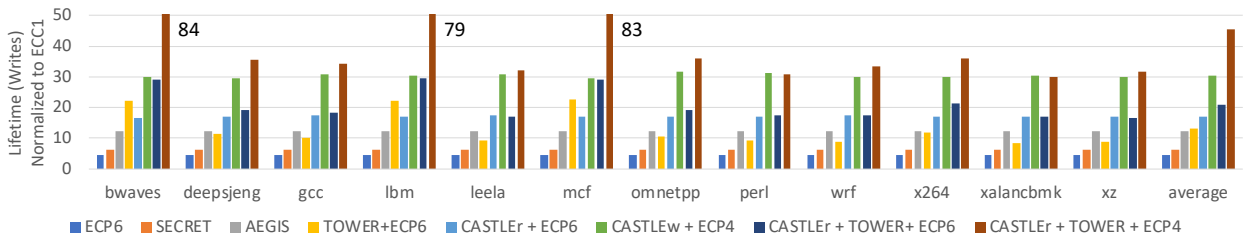


Figure 19: Lifetime (writes until failure) for an 1MB memory with mean cell lifetime of 10^8 writes, with 0.2 CoV. These results assume perfect wear-leveling. Note that all CASTLE and TOWER results shown here, like SECRET, include subcounters to minimize writes to clean words. SECRET bar includes ECP-6.

protected system and ECC-1, respectively. On average across the representative SPEC2017 benchmarks, this improvement for CASTLEw and TOWERS over ECC-1, SECRET, and AEGIS is 45×, 7.1×, and 3.7×, respectively, providing a 1.5× improvement over CASTLEw alone.

Figure 20 presents a sensitivity study of the lifetime improvements of CASTLE+TOWER over increasing coefficient of variation and different area overhead. The height of the orange bars represent the additional writes the schemes enable before failure, while the line chart represents improvement over iso-area SECRET (the ratio of the orange bars over the blue bars). Looking at the figure, increasing the CoV decreases the overall lifetime and the raw improvements over SECRET alone, while raising the relative improvement over SECRET. This indicates that while the benefits of both SECRET and our proposed schemes deteriorate in lifetime as CoV increases, CASTLE and TOWERs deteriorate more slowly than SECRET, gaining increased relative lifetime as variation increases. At 12% area overhead, the relative improvement over iso-area SECRET for CASTLEw+TOWER increases from 7.1× to 8.2× (0.25 CoV) and 9.8× (0.3 CoV). However, for systems with more constrained area overhead (8% area), this improvement range can be as high as from 8.5× (0.2 CoV) to 13.8× (0.3 CoV) compared to iso-area SECRET.

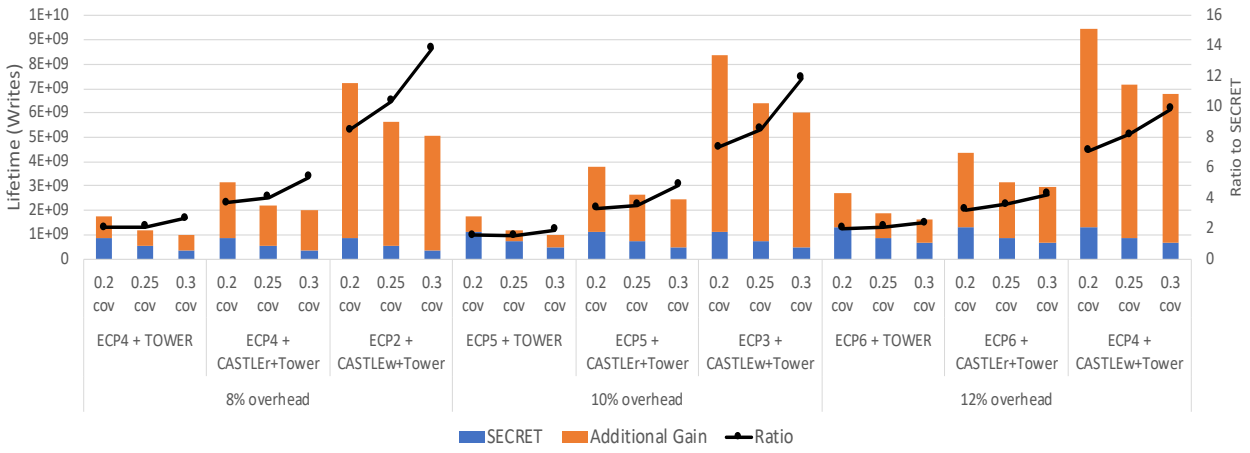


Figure 20: Lifetime added to iso-area ECP (stacked bar), as well as the ratio of lifetime to iso-area ECP (lines) for variable cov and area overhead (SPEC2017). Note that all CASTLE and TOWER results shown here, like SECRET, include subcounters to minimize writes to clean words.

3.4.6 Performance Study

The IPC results for counter-mode encryption using the baseline of SECRET are shown in Figure 21. At 10^{-4} weak cell rate, using CASTLE is rarely necessary, and thus the IPC impact is negligible. Adding TOWER to CASTLE at 10^{-4} weak cell rate reduces the IPC on average by 0.53%, but provides gains in endurance and energy as discussed in previous sections.

Moving to the 10^{-2} weak cell rate, the counters must be advanced more frequently, and as a result the IPC of CASTLE reduces by 0.42%. Adding TOWERs further reduces the IPC, for a total of 0.72% on average. The worst-case IPC degradation of CASTLE+TOWER at a 10^{-2} rate is less than 3%, and this fault rate only occurs at the end of life, demonstrating that the performance gracefully degrades as the memory becomes more faulty. One noteable result is for the benchmark xalancbmk, which is the only benchmark with CASTLE having a lower IPC than CASTLE+TOWER. This is due to the higher number of advancements required on average, which penalizes IPC more evaluating coset candidates.

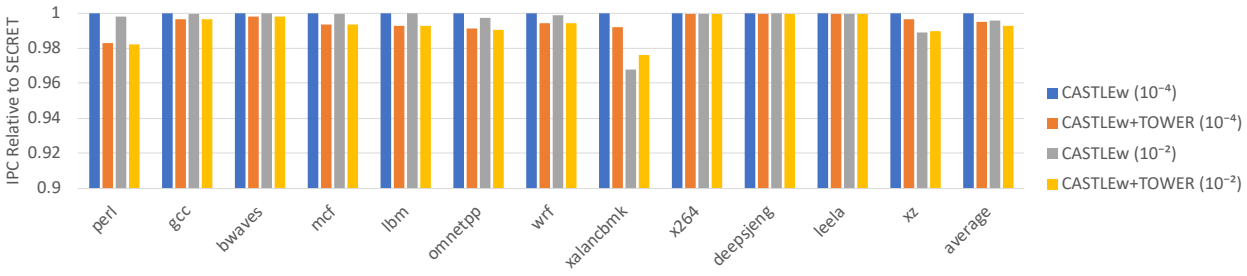


Figure 21: IPC relative to a system using SECRET for various configurations of CASTLEw with ECP-4, with and without TOWER.

We also conducted a sensitivity study across different devices using features sizes from 180nm down to 28nm. These results are shown in Figure 22 for CASTLEw+TOWER at a fault rate of 10^{-4} normalized to SECRET. Across the different feature sizes, we see little change in the overall IPC, indicating that the scale of change in write latencies across feature sizes is not appreciably different and the variation from an IPC of 1 is largely due to noise in the simulator. The only benchmark which did not follow this trend was xz. This benchmark is particularly sensitive to variations in the miss rate of the L1 TLB, and is among the poorest performers in branch prediction [52]. Over several trials, the L1 miss rate varied by as much as 10× between the CASTLEw + TOWER and SECRET performance, amplifying the latency of write operations beyond the relatively small changes associated with the different protection techniques.

In summary, CASTLE leverages the nature of block cipher encryption to improve reliability for systems that use in-memory encryption in memory with endurance limitations that manifest as stuck-at values. In particular, CASTLE provides 30×, 4.7×, and 2.4× longer lifetime than SECDED ECC; SECRET, which uses six ECP pointers; and AEGIS, which has a correction guarantee of 11 faults, respectively. CASTLE can also be used to reduce error correction overhead, achieving a similar protection as five ECP pointers with only a single pointer at moderate fault rates. Furthermore, CASTLE can maintain an UBER of at least 10^{-11} with the same error correction as the leading related work, SECRET [38], for extremely high fault rates of 10^{-2}. In contrast, SECRET can only achieve an UBER of 10^{-6}.

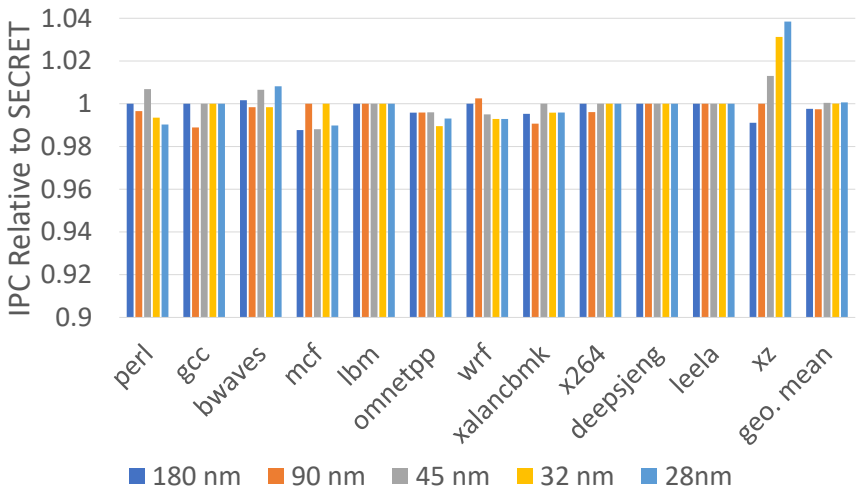


Figure 22: IPC of CASTLEw+TOWER at 10^{-4} fault rate relative to SECRET for various feature sizes.

TOWERs is, to our knowledge, the first scheme providing lightweight compression on encrypted data. Using the reclaimed bits from compression, TOWERs applies a coset on the encrypted data to further improve fault tolerance while also improving energy and endurance. TOWERs provides an additional lifetime benefit of 1.5× over CASTLE, which can reach 2.9× improvement for highly compressible benchmarks. Across the SPEC 2017 workloads, TOWERs reduces energy over SECRET by about 14%, which results in a cumulative improvement in energy of 54% over an encrypted baseline, including one protected by a partition-and-flip fault tolerance technique, such as AEGIS.

In the next chapter, I present my work on a similar approach to fault tolerance and energy reduction in MACE WINDU. These techniques are specifically tuned to PCM using AES-XTS encryption, and serve as an illustrative example of how lifetime energy consumption can be assessed and analyzed to guide memory system design.

4.0 MACE WINDU

"Memory AES-XTS Cosets with Energy-efficiency" (MACE) was designed to extend the lifetime of PCM with in-memory encryption using cosets to generate alternate candidates for writebacks. Using a similar approach to that of TOWERs, MACE can mask faults associated with stuck bits, and opportunistically reduce write energy as a secondary optimization parameter. As with CASTLE & TOWERs, MACE is complementary with existing error correction techniques, and provides opportunities to finely tune the tradeoff of cosets and error correction to meet the needs of the system at hand. In support of MACE, I present "Word-based, Interleaved, Nonessential-bit Decomposition Utility" (WINDU) which leverages lightweight compression in order to store MACE auxiliary bits in reclaimed space.

4.1 MACE Encoding

The leading fault tolerance approaches for in-memory encryption operate in concert with ECP. SECRET [38] adds additional auxiliary capacity to store ECP pointers directly while counter advance (used for CASTLE) uses the AES block-cipher itself to generate cosets on demand when the correction capability is exceeded [7]. However, this approach relies on the fact that the counter used in AES-GCM and CASTLE will produce a new random value every time; for unbiased data, a random coset will be more effective than a fixed or biased one [47, 34]. In contrast, AES-XTS will generate its OTP from an initial value based on the address, but that value is invariant for the block being encrypted. Thus, the leading approach in CASTLE is not a practical approach for mitigating endurance faults. Thus we fall back to the prior approach of appending auxiliary space to the end of each memory row for ECP as shown in Figure 23.

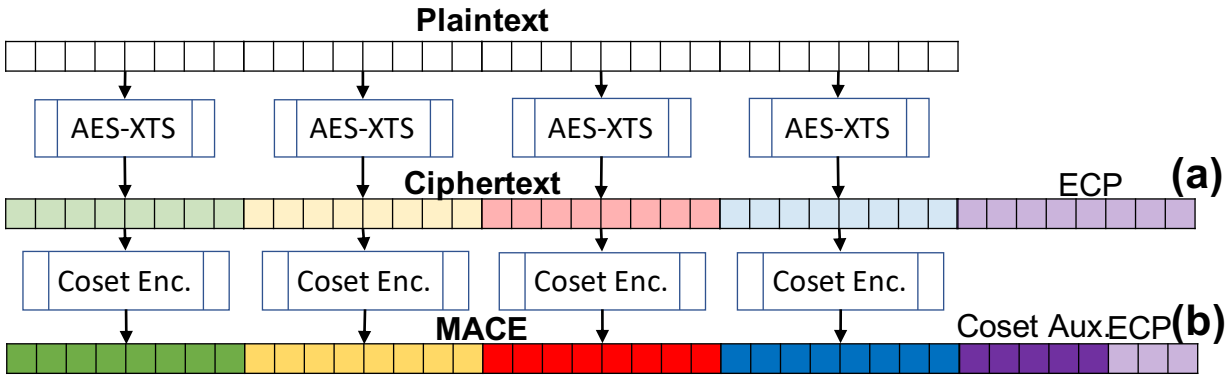


Figure 23: AES-XTS encryption with (a) ECP fault tolerance and (b) MACE fault tolerance.

MACE leverages the idea that random cosets can be effective to optimize codewords for a particular criterion [47, 34] including minimizing SA-W data [64, 33]. MACE, shown in Figure 23(b), takes the ciphertext and conducts coset encoding for reliability using a set of randomly generated coset candidates [47]. The binary ID of the coset candidate that minimizes SA-Ws is stored as auxiliary information. To retain a near-iso-area overhead, coset auxiliary bits (dark purple) may replace bits that would have been allocated for ECP (light purple). Any remaining SA-W bits after coset encoding are corrected by the remaining ECP. Thus an acceptable writeback candidate will reduce the number of SA-Ws such that pointer fault tolerance can handle them.

We see from Figure 23 that cosets are applied locally to blocks smaller than the row, but pointers are global for the row. To maximize fault tolerance it is desirable to locally minimize SA-Ws in each word. As discussed in Section 2.3, in-memory encryption defeats locality-based techniques to reduce bit changes. However, early in the memory lifetime, there is low pressure on fault tolerance. Thus, MACE can apply a secondary optimization metric such as flip minimization, improving energy and endurance.

MACE uses the same approach as TOWERs to apply a multi-objective cost function for coset selection. Assuming some method of distinguishing SA-R and SA-W faults, there may be several candidates which minimize errors, which can then be optimized according to a secondary metric such as energy reduction. As with CASTLE and SECRET, we assume

that writes are only committed for dirty words. However, if all the available candidates fail to reduce the number of faults to the correction capacity, the entire row can be re-encoded by the best coset to potentially free another pointer for use in the word being written. This progressively more conservative approach allows Algorithm 1 to opportunistically reduce the secondary cost while avoiding taking on more SA-W bits than the correction scheme can support.

Coset effectiveness improves as the codes are applied to smaller sub-row blocks or the number of codes per block increases, both at the expense of increased auxiliary bits. TOWERs and previous work have explored compression to create space for auxiliary bits within the data block to mitigate this overhead [46, 48]. AES-XTS's block-cipher encryption of data makes efficiently storing auxiliary data in the space reclaimed by compression apparently intractable; for counter-mode-based encryption the bits reclaimed by compression can simply be overwritten because the encryption is applied via `XOR` with the OTP. Unfortunately, AES-XTS introduces a block cipher encryption between the OTP steps, diffusing information from every bit of the input to every bit of the output. Rather than simply pay the penalty of reserving additional capacity for coset ID encoding, I developed WINDU.

4.2 WINDU Architecture

Applying lightweight compression to reclaim space for storing auxiliary bits representing the coset encoding is non-trivial for AES-XTS. In MACE, cosets are applied *after* encryption due to the simple `XOR` function required to generate each write candidate. Lightweight compression must be completed *prior* to encryption as it leverages similarity in the data, which is destroyed by the encryption process. Thus, if the coset can be applied prior to encryption, each the coset candidate can be generated from the compressed word using the `XOR` function and concatenated with the auxiliary value, which is stored in the reclaimed bits. While this is practical for a OTP, it is impractical for AES-XTS, because each candidate must be encrypted separately with an expensive block-cipher operation.

Thus, to leverage reclaimed space from lightweight compression for storing MACE auxiliary information, we propose the Word-based Interleaved Nonessential-bit Decomposition Utility (WINDU), shown in Figure 24. MTC lightweight compression [48] is applied to each word in the memory row to recover 5-7 bits per word, shown as white boxes. As with TOWERs, MTC techniques are selected for their simplicity and compression coverage; matching bits in the leading byte of each word are readily identified and apply to a large proportion of data. The compressed data is then repacked toward the MSB, to absorb the reclaimed space and clustering the unused bits at the end of the row. AES-XTS then proceeds as in Figure 23 with the exception of the tail of the blue data and the reclaimed space. After the MACE cosets are determined for the encrypted data, their auxiliary encoding is added to the reclaimed space of the last block, encrypted, and then passed through the coset function with auxiliary data stored externally.

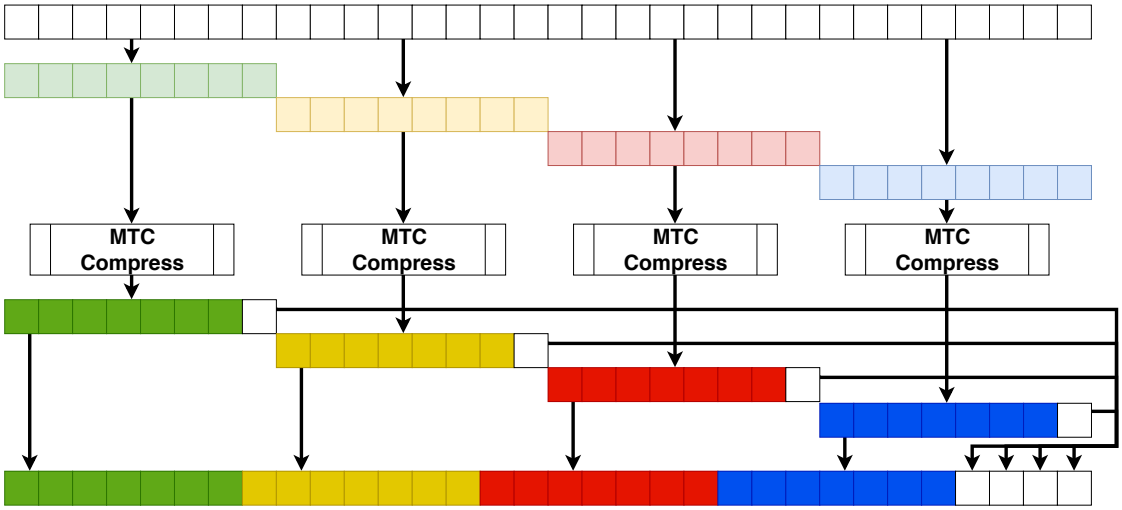


Figure 24: Applying WINDU to a cache line before writeback. Each word in the line is compressed, the compressed data is packed into the MSBs of the cache line, and then each word in the resultant line is assessed to determine the best coset to apply.

This approach requires only $\log_2(n) + 2$ bits to encode the coset codeword, where n is the number of coset candidates and the remaining two bits indicate one of three MTC compression states for the row ("01," "10," or "11") or that the row is uncompressed ("00"). For 512-bit rows, 64-bit words, and 32-bits cosets per word, the encoding bits will only

consume 35 of the 40 bits reclaimed by compression. If the seven external auxiliary bits are protected with ECP-1, this is adds four additional bits (three address bits plus one replacement bit), requiring a total of 11 bits, which is 2.2% of the total row as overhead. Thus, MACE WINDU essentially repurposes the bits of one pointer for this encoding. Compared to MACE, this a significant area advantage, as 32 cosets would require reclaiming four pointers.

We can improve on this overhead by storing the compression bits in the reclaimed space. We can also leverage parallel AES units and *multi-stage* coset encoding to reduce this further. For a 512-bit row with 128-bit encryption blocks, the system needs at least four AES units. Using these units in parallel, we can create four *candidate parents* (first stage) by encoding pre-encryption candidate with a unique two-bit value in the remaining reclaimed bits. This will generate four uniquely encrypted words. Using eight vector MACE post-encryption cosets on each parent candidate (second stage) generates 32 unique candidates, while needing to store only the single three-bit ID of the coset applied. With eight parallel AES units, the reclaimed bits are entirely filled and only two external bits are needed. These remaining $\leq$3 bits can be protected through triple-modular redundancy, requiring a total of 9 (or 6) auxiliary bits (1.8% or 1.2% overhead) and reclaiming only 1 pointer.

There are performance ramifications for not enforcing word alignment in memory. For all but the most significant word (MSW), the compressed data must be packed such that it will cross a word boundary. Since all the encoding for the MSWs is stored in the LSW, every access requires decoding and decrypting the final word. In the worst case, we need to access three words for every transaction: the words on either side of the boundary the compressed data crosses, and the final word to retrieve the encoding information. For memory systems which employ critical word first and early restart, applying WINDU will incur this additional access penalty. For a 512-bit row size using 64-bit words, assuming that the word accesses are evenly distributed, we expect that transactions on the first and last word will require accessing two words and for the remaining words accessing three words, with an expected value of $2 \cdot \frac{2}{8} + 3 \cdot \frac{6}{8} = 2.75$ word accesses per word transaction. Another potential performance issue is the latency associated with serializing the encryption of the final word. This roughly doubles the latency associated with encryption. However, WINDU provides comparable coset protection while reclaiming three fewer pointers than MACE alone.

4.3 LARS Sustainability Analysis

Prior work in GreenChip [65] developed the indifference analysis shown in Equation 3 where M_i is the embodied (*i.e.,* manufacturing) energy and P_i is the operational power of system i. It is sufficiently flexible for different holistic sustainability evaluations, when system reliability is constant between design choices. However, when evaluating an approach like MACE, different configurations will have an impact on lifetime due to memory wearout—different points in the tradespace of using auxiliary space for ECP versus MACE will impact dynamic energy and cumulative energy consumed by the system. If a system has lower use-phase energy and lower embodied energy, but it must be replaced every month, this replacement embodied energy should be taken into account when comparing it with another system. Thus, we proposed an extension to indifference theory called "Lifetime Amortized Replacement for Servers" (LARS) indifference analysis.

$$t_I = \frac{M_1 - M_0}{P_0 - P_1} \tag{3}$$

$$A_i = \frac{M_i}{L_i} = \frac{M_i \cdot W_i}{WBF_i} \tag{4}$$

$$t_{I_{LARS}} = \frac{M_1 - M_0}{(P_0 + A_0) - (P_1 + A_1)} \tag{5}$$

The fundamental difference between LARS indifference analysis and prior sustainability indifference analysis [65] is the inclusion of amortized embodied energy of replacements reflected in Eq. 5. In Eq. 3, the indifference point reports the time it takes for a system with lower operational power to save the equivalent energy of the larger embodied energy from more complicated manufacturing. However, when the mean-time-to-failure (MTTF) interval lapses for one system prior to the indifference time, traditional analysis is less meaningful. With LARS indifference analysis, we consider replacement embodied energy as a cost per time shown in Eq. 4, in a similar fashion to operational power. A_i is the embodied energy, M_i, divided by the lifetime, L_i, for a system i. L_i can be determined as the ratio of writes before failure WBF_i to the write velocity (writes per second) W_i of system i. The LARS

comparison assumes two systems under comparison operate until failure, are replaced with the same system, and the cycle continues indefinitely. Essentially, the LARS indifference time now considers replacement cycle along with embodied energy, operational energy, and usage scenario.

4.4 Experimental Setup for MACE WINDU

To evaluate MACE and MACE WINDU, we created a modified Pintool [50] to monitor all memory writes (cache writebacks) encountered at the lowest-level, unified cache. The provided cache monitoring instrumentation was adapted to extract the address and data of 512-bit cache lines as they were evicted. This instrumentation was attached to execution runs of a selection of the SPEC2017 benchmarks. The benchmarks listed in Table 1 were selected for their representative coverage of the rest of the suite [52] and augmented with any benchmarks which exhibited a relatively high percentage of memory operations. All benchmarks were compiled using the SPEC base and speed metrics to incur higher memory use, and executed with reference workloads.

As with CASTLE & TOWERs, to evaluate reliability, fault maps were generated to match incidence rates of $10^{-5}...10^{-2}$, representing high to extreme cases of cell failure. Each of the fault maps specifies permanently failed bits as stuck-at '0' or stuck-at '1,' identified by their bitwise row and column position in a 2GB memory. A Bayesian distribution was used to approximate the spatial correlation of faults [25]. Under simulation, the fault maps were used to generate fault vectors for each cache line writeback based on the trace address of the benchmark under test. For lifetimes, cells were assumed to have an average of 10^8 write cycles before failure following a standard distribution [66, 32, 58] with a pessimistic coefficient of variation (CoV) of 0.25 [32, 58] and perfectly uniform wear-leveling [58] consistent with many proposals in the literature [59, 60, 58]. Experimental results represent an average over 10 experiments.

To evaluate the overall performance impact of MACE WINDU in the context of AES-XTS protection, we used the SNIPER full-system simulator [62] to monitor performance for each

of the selected benchmarks. The simulation system parameters are shown identical to those used in the evaluation of CASTLE & TOWERs as shown in Table 2. For this experiment, it was assumed that a parallel hardware implementation of AES-XTS was used, and that the latency of AES-XTS encryption is double that of an AES block-cipher. To model the latency of coset encoding/decoding, we used parameters from a hardware simulation presented in previous work on coset encoding [46].

4.5 Results

Based on the experimental setup described in Section 4.4, we conducted detailed evaluations of MACE and MACE WINDU with various configurations of external auxiliary bits and ECP to understand the impact of our contributions.

4.5.1 Compressibility and Coset Cardinality

MACE WINDU can only be applied when all the words within each cache line are compressible. Thus, we explored compression coverage, measured as the ratio of compressible cache lines written to all cache lines written, for a modified version of MTC [48]. As with our evaluation for the CASTLE & TOWERs work, we found that most workloads exhibited high compression coverage, especially after an initial warm-up period. This study is identical to that presented in Section 3.4.2, shown in Figure 14.

To guide the selection of the cardinality of the coset, denoted with m bits to generate 2^m candidates, we studied $m \in \{2, 3, ..., 7\}$. The coset included $2^m - 1$ pseudo random vectors and the identity vector. Using a fault incidence rate of 10^{-2}, A representative subset of 1M cache line writebacks was evaluated for each benchmark, and arithmetic mean of the total SA-W bits encountered was recorded. The results indicate a diminishing return for more than 32 cosets ($m = 5$), with the knee of the curve between 16 and 32 cosets. Given the amount of bits reclaimed per word by MTC of five bits, we conclude that 32 cosets is the ideal cardinality in terms of efficacy of MACE and efficient use of reclaimed bit area for

WINDU, and assume this number of cosets for the remaining experiments. This study is identical to that presented in Section 3.4.3, shown in Figure 14.

4.5.2 Reliability and Lifetime

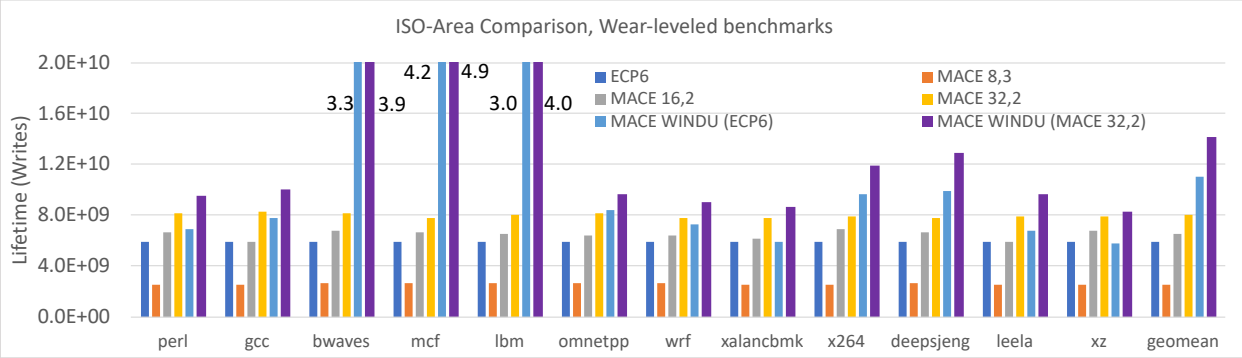


Figure 25: Mean lifetime writes for various combinations of ECP-N, MACE, and WINDU applied to the benchmarks under test. Assuming iso-area configuration, we compare ECP-6, MACE alone with 8, 16, and 32 cosets, and WINDU with either ECP-5 or MACE. For the latter, WINDU is applied to compressible lines, and either ECP-6 or ECP-2 + MACE otherwise.

To evaluate the effect of MACE on PCM lifetime, iso-area configurations of MACE and MACE WINDU were compared to the baseline of ECP-6 [38] when protecting a simulated memory system to failure. Each experiment continually issued writes until the simulation recorded four cache line writebacks with unrecoverable bit errors [59].

Figure 25 reports the results. MACE with eight cosets and three pointers (MACE 8,3)[1,2] performs poorly, but increasing to 16 cosets while losing one pointer (MACE 16,2)[1,2] slightly outperforms ECP. With evenly distributed faults, the probability of failure is similar for both approaches. However, the spatial correlation of faults sees more instances where there are more faults than ECP can correct, and the additional cosets outperform the ECP.

[1]Coset auxiliary bits are protected with a single ECP-style pointer.

[2]An uncorrected stuck-at failure in the coset bits does not result in a failure, but reduces the number of coset candidates by half, decreasing effectiveness.

Increasing to 32 cosets is the best MACE-only configuration as it retains the same number of pointers (MACE 32,2)[2]. Thus, from here on, MACE assumes MACE 32,2 unless specified. MACE WINDU, which reverts to ECP-6 when no compression is possible improves over MACE on average, but this improvement is not consistent for all applications. However, when MACE WINDU reverts to MACE without compression, there is improvement for all compressible benchmarks. MACE WINDU with ECP or MACE 32,2 provides an average lifetime improvement of 2× and 2.6× over ECP, respectively and 40% and 81%, respectively over MACE. This shows that MACE WINDU can improve fault tolerance over MACE in an iso-area setting.

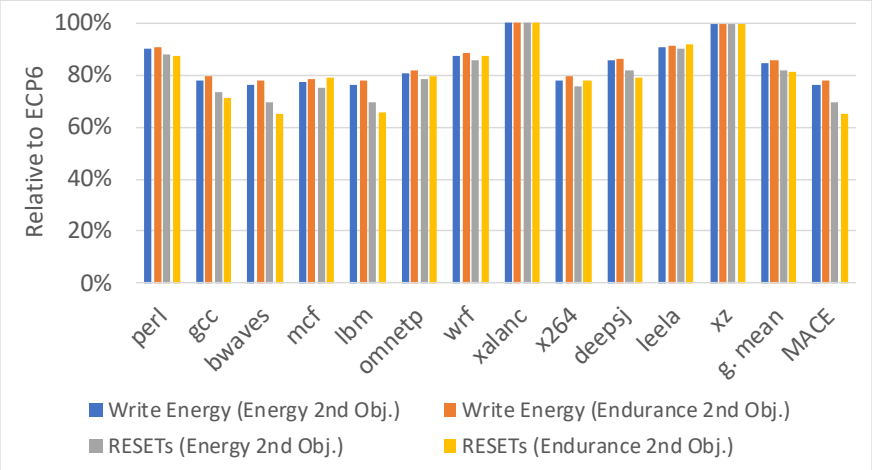


Figure 26: Improvement in differential write energy and endurance (RESETs) relative to ECP. MACE 32,2, invariant of compressibility, summarized with a single entry. Incidence fault rate of 10^{-2}.

4.5.3 Energy and Endurance

To evaluate the impact of the MACE approach we conducted an energy and endurance evaluation, including the bit changes associated with writing auxiliary bits. Figure 26 depicts the improvement in differential write energy and count of RESET operations, expressed as a percentage, normalized to ECP only (control) for each benchmark using MACE WINDU, assuming a fault incidence rate of 10^{-2}. A similar calculation was done for MACE. As

MACE is invariant to compression, the energy and endurance calculation is similar for each benchmark and is summarized in the single MACE entry in the figure.

As expected, MACE WINDU improvement tracks workload compressibility. Furthermore, there is minimal distinction between optimizing energy or endurance, as both benefit from minimizing bit changes. MACE improves over MACE WINDU in cases where the compressibility is moderate to low but has a disadvantage in terms of storage overhead. We explore this further in Section 4.5.5.

4.5.4 Performance Impact

To evaluate the impact of MACE WINDU on performance, IPC was measured for AES-XTS+ECP versus AES-XTS+MACE WINDU. Note, AES-XTS+MACE is similar to AES-XTS+ECP because MACE does not significantly impact the read critical path delay like MACE WINDU. Figure 27 shows the results, normalized to AES-XTS+ECP, using a fault incidence rate of 10^{-2}. As expected, we can see a general trend that more compressible workloads tend to have reduced IPC, owing to the additional overhead for applying and evaluating coset candidates. This pattern is perturbed somewhat by the proportion of memory instructions in each workload. However, the performance penalty never exceeds the energy saved (Figure 26), so the additional runtime for any given task will still save overall memory energy.

4.5.5 LARS Whole-lifetime Energy Analysis

Based on the results of the lifetime and energy studies, a sustainability analysis was conducted using a modified version of the GreenChip [65] tool to include LARS indifference analysis. The observed lifetimes for each of the configurations were used alongside estimates of both embodied and operational energy consumption to evaluate the lifetime energy footprint.

Figures 28a, 28b, 28c show indifference point plots, calculated using Eq. 3 for (a) ECP-6 versus MACE 32,6 (32 cosets, 6 pointers), (b) MACE WINDU (MW), which reverts to MACE 32,2 when compression is not possible, versus MACE 32,6 for a 1TB memory, and

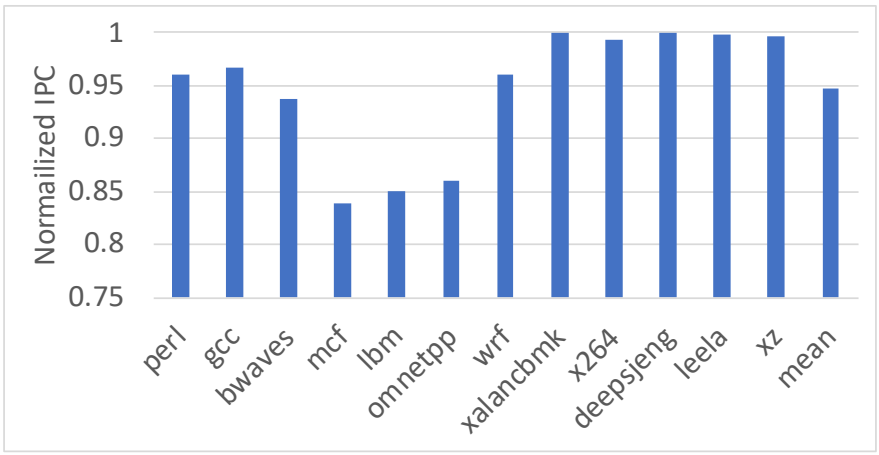
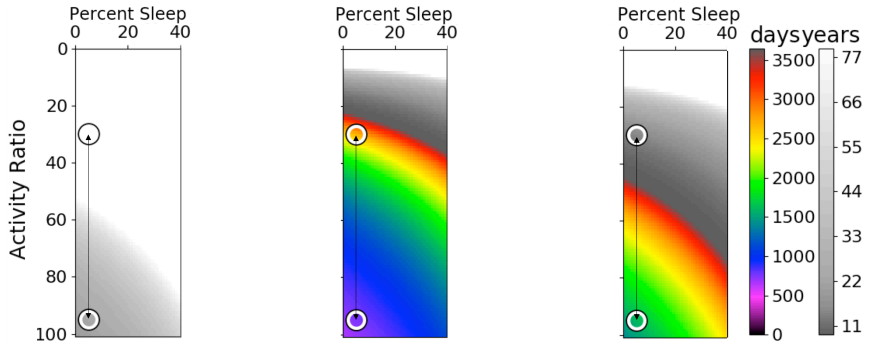


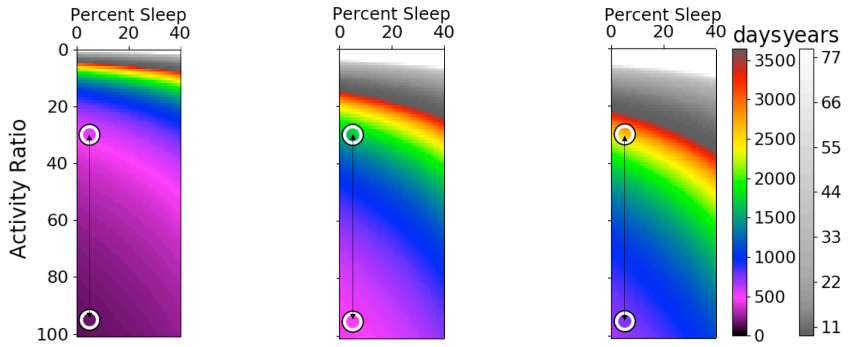
Figure 27: Simulated IPC for MACE WINDU, normalized to ECP-6, using a fault incidence rate of 10^{-2}.



(a) ECP vs. MACE (b) MW vs. MACE (c) MW vs. MACE (4TB)

Figure 28: Indifference points (t_I) for the traditional GreenChip tool for ECP-6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted.

(c) the same MW versus MACE comparison for a 4TB memory. Note, the MACE configuration consistently has a larger area overhead, *i.e.,* embodied energy, with a lower operational energy. The figures highlight a range of usage scenarios for blade servers, from from 95% uptime and low utilization (underloaded cloud) to similar uptime and high utilization (heavily loaded cloud or supercomputer).



(a) ECP vs. MACE (b) MW vs. MACE (c) MW vs. MACE (4TB)

Figure 29: LARS indifference points (t_I) for the GreenChip tool for ECP-6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted.

Figure 28a indicates that MACE requires ≥29 years to recoup the additional embodied costs from the area overheads. Similarly, Figures 28b and 28c show MACE requires 2.1-7.3 years and 4.3-17.5 years, reflecting highest to lowest activity, for 1TB and 4TB memories, respectively.

Recalling from Section 4.5.5, that Eq. 3 does not accurately reflect indifference when reliability and replacement cycle are a function of the system comparison. Figure 29 shows the same comparison as Figure 28, except using Eq. 5 which takes into account amortized embodied energy from replacements (Eq. 4). The biggest change is in Figure 29a, where indifference time of 29 years drops to <5 months, and low utilization approaching ∞ becomes <18 months. The MW versus MACE comparisons show similar adjustments where Figures 29b and 29c report 1.3-4.5 years and 2.0-7.1 years, respectively, reflecting approximately 60% and 2× lower indifference time for 1TB and 4TB memories, respectively. Clearly, from Figure 29, MACE is the considerably more sustainable choice than ECP, but the choice of MACE WINDU versus MACE is more dependent on system configurations. MACE WINDU is more sustainable for moderate to large memories with MACE being more attractive for very large memories. This suggests a tradeoff between total capacity, effective capacity (total capacity less correction overhead), and activity ratio of a server.

Together, MACE and WINDU demonstrate improved lifetime and reduced energy consumption when applied to encrypted non-volatile memories. MACE readily generates write-back candidates with fewer SA-W faults, and opportunistically reduces dynamic energy by 15% on average. The addition of WINDU allows the technique to be applied within the existing error correction overhead, to achieve upwards of 2.6× improvement in lifetime over ECP-6. The LARS indifference analysis extends the indifference analysis of GreenChip to include the memory lifetime. This permits a holistic study of the tradespace between activity, capacity, and correction overhead in the context of non-volatile memories.

In the final chapter, I present a summary of my contributions to date and how my body of work supports the goals of this thesis, alongside directions for future study.

5.0 Conclusion & Future Work

CASTLE and TOWERs [7], of which TOWERs and the simulation/evaluation work are my primary contributions, can achieve an average lifetime improvement of over 45× compared to SECDED ECC, 7.1× compared to SECRET, and 3.6× compared to the leading partition-and-flip fault-tolerance approach (AEGIS) for the same area overhead. CASTLE leverages the counter-mode encryption process to improve reliability in the presence of endurance faults. TOWERs for CASTLE improve reliability as well as energy for encrypted data through a novel application of compression and encoding. CASTLE and TOWERs are compatible with error-correction codes (ECC) and error correction pointers (ECP), the standard for mitigating endurance faults in PCM. Together, they represent an immediately applicable technique which meets the goals of this thesis using the state-of-the-art PCM technology.

I designed MACE and WINDU [8] for PCM in a high-capacity cloud computing context, for which AES-XTS encryption has been established as the *de facto* standard for data confidentiality. In this setting, MACE improves the lifetime and dynamic energy of PCM by up to 2.6× and 15%, respectively. The WINDU architecture leverages lightweight in-memory compression akin to TOWERs to reduce the auxliary overhead of the MACE technique. The addition of WINDU allows the technique to be applied within the existing error correction overhead—I have shown that MACE WINDU can achieve better fault tolerance and endurance than ECP alone for the same area overhead. For workloads that exhibit low compressibility, MACE makes more effective use of overhead than ECP, and secures an energy benefit. To more thoroughly explore the cradle-to-grave energy consumption of a system, which is especially salient to system designers of cloud computing servers, I co-designed the LARS indifference analysis approach. This holistic view of lifetime energy consumption accounts for replacement of failed memory devices informs system design and memory selection. The LARS indifference analysis demonstrates that the endurance and energy benefits of MACE and MACE-WINDU are realized on a relatively short time scale, and illustrates situations where server utilization and total storage capacity favor one or the other.

In future work, all of the above techniques could be evaluated in concert with other fault tolerance techniques. The fault tolerance benefit of my work is agnostic to the secondary fault tolerance approach with which they are combined. Although this study uses ECP, which was inherited by building upon SECRET, in principle, CASTLE and TOWERs or MACE can be combined with a partition-and-flip fault tolerance approach like AEGIS, which could increase the lifetime further. Similar studies of iso-area lifetime and energy improvements could compare any such fault mitigation scheme.

Using AEGIS with SECRET, and by extension word level CASTLE and TOWERs, creates an undesirable side effect. While ECP correction can be localized to the encrypted word, AEGIS requires bit changes throughout the row, disrupting SECRET's benefit of only writing dirty words. New faults often require AEGIS to restore flipped groups of the current partition and flip bits of a new partition, impacting SECRET's energy savings. TOWERs could partially mitigate this overhead, by extending the coset selection heuristic (Algorithm 1) beyond the word to the entire row. We hope to study this in future work. Of course, the row version of CASTLE and TOWER can use AEGIS without these concerns. Moreover, iso-area word-level CASTLE and TOWERs (ECP-6) improves lifetime by 3.7× with a 54% energy reduction compared to AEGIS, showing the efficacy of the CASTLE and TOWERs approach.

With continued process development and scaling, the access time for PCM is expected to improve. Beyond process maturation, novel cell designs, material selection, and alternate access circuits (*e.g.,* optically-switched PCM films [67], carbon nanotube access [14]) are promising approaches to reducing the relatively high write latency of PCM cells. In the interim, my work could incorporate additional techniques to improve the access latency at the memory controller and cache level. My work to date assumes the the memory controller and how it accesses the underlying PCM is opaque to the cache and other upstream units. With specific knowledge of what data are dirty, the load on the memory transaction queues, and the parameters of the encryption process, there is an opportunity to improve the overall average memory access time of a PCM main memory unit. This is a departure from the DRAM-compatible designs in the literature to date, but such work is a necessary step to establishing PCM as a standalone main memory.

The continued demand for denser, faster, and more energy-efficient memory systems is driving research of emerging memory technologies to overcome the limitations of DRAM. Already, PCM is emerging as a potential unified memory which can act as both main memory and storage-class memory. However, its outstanding challenges in reliability, energy efficiency, and security must be resolved before PCM is a viable candidate. In my body of work, I have drawn attention these three inseparable design concerns as the most pressing challenges to overcome. This thesis presents two approaches which address these concerns simultaneously, prepared for the context of standardized in-memory encryption and evaluated under representative modern workloads. Together, these techniques advance the state-of-the-art and represent a step toward the adoption of PCM as a main memory technology.

Bibliography

[1] L. Johnsson and G. Netzer, "The impact of Moore's Law and loss of Dennard scaling: Are DSP SoCs an energy efficient alternative to x86 SoCs?," *Journal of Physics: Conference Series*, Vol. 762, p. 012022, oct 2016.

[2] O. Villa, D. R. Johnson, M. Oconnor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, *et al.*, "Scaling the power wall: a path to exascale," *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 830–841, IEEE, 2014.

[3] S. A. McKee, "Reflections on the memory wall," *Proceedings of the 1st conference on Computing frontiers*, p. 162, 2004.

[4] Intel Corporation, "Product Brief: Achieve Greater Insight from Your Data with Intel Optane Persistent Memory," Tech. Rep., Intel Corporation, March 2021.

[5] J. Choe, "Intel 3D XPoint Memory Die Removed from Intel Optane™ PCM (Phase Change Memory)."

[6] Intel Corporation, "336907-002US: Memory Encryption Technologies Specification," Tech. Rep., Intel Corporation, Santa Clara, CA, United States, 2019.

[7] S. Longofono, D. Kline, R. G. Melhem, and A. K. Jones, "A CASTLE with TOWERs for Reliable, Secure PCM," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[8] S. Longofono, D. Kline, R. Melhem, and A. K. Jones, "Toward Secure, Reliable, and Energy Efficient Phase-change Main Memory with MACE," *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8, 2019.

[9] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced Encryption Standard (AES)," 2001-11-26 2001.

[10] D. Ielmini, D. Sharma, S. Lavizzari, and A. L. Lacaita, "Reliability Impact of Chalcogenide-Structure Relaxation in Phase-Change Memory (PCM) Cells—Part I: Experimental Study," *IEEE Transactions on Electron Devices*, Vol. 56, No. 5, No. 5, pp. 1070–1077, 2009.

[11] W. Zhang and T. Li, "Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system," *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pp. 197–208, 2011.

[12] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proceedings of the IEEE*, Vol. 98, No. 12, pp. 2201–2227, Dec 2010.

[13] S. Yoo, H. D. Lee, S. Lee, H. Choi, and T. Kim, "Electro-Thermal Model for Thermal Disturbance in Cross-Point Phase-Change Memory," *IEEE Transactions on Electron Devices*, Vol. 67, No. 4, No. 4, pp. 1454–1459, 2020.

[14] F. Xiong, *Scaling study of phase change memory using carbon nanotube electrodes*. PhD thesis, University of Illinois, Urbana-Champaign, IL, 2014.

[15] Kinarn Kim and Su Jin Ahn, "Reliability investigations for manufacturable high density PRAM," *2005 IEEE International Reliability Physics Symposium, 2005. Proceedings. 43rd Annual.*, pp. 157–162, 2005.

[16] S. Lee, J. Jeong, T. S. Lee, W. M. Kim, and B. Cheong, "A Study on the Failure Mechanism of a Phase-Change Memory in Write/Erase Cycling," *IEEE Electron Device Letters*, Vol. 30, No. 5, No. 5, pp. 448–450, 2009.

[17] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), p. 14–23, Association for Computing Machinery, 2009.

[18] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, Vol. 29, No. 2, No. 2, pp. 147–160, 1950.

[19] S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 8, No. 2, No. 2, pp. 300–304, 1960.

[20] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, Vol. 3, No. 1, No. 1, pp. 68–79, 1960.

[21] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, Vol. 8, No. 1, No. 1, pp. 21–28, 1962.

[22] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *ACM SIGARCH Computer Architecture News*, Vol. 38-3, pp. 141–152, ACM, 2010.

[23] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pp. 323–326, Dec 2005.

[24] T. Yuan, S. Z. Ramadan, and S. J. Bae, "Yield prediction for integrated circuits manufacturing through hierarchical Bayesian modeling of spatial defects," *Transactions on Reliability 2011*, Vol. 60, No. 4, No. 4, pp. 729–741, 2011.

[25] Z. Al Ars, *DRAM fault analysis and test generation.* TU Delft, Delft University of Technology, 2005.

[26] D. Kline, R. Melhem, and A. K. Jones, "Sustainable fault management and error correction for next-generation main memories," *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–6, 2017.

[27] D. Kline, J. Zhang, R. Melhem, and A. K. Jones, "FLOWER and FaME: A Low Overhead Bit-Level Fault-map and Fault-Tolerance Approach for Deeply Scaled Memories," *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 356–368, 2020.

[28] S. Longofono, D. Kline, R. Melhem, and A. K. Jones, "Predicting and mitigating single-event upsets in DRAM using HOTH," *Microelectronics Reliability*, Vol. 117, p. 114024, 2021.

[29] T. M. Hollis, "Data Bus Inversion in High-Speed Memory Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 56, No. 4, No. 4, pp. 300–304, 2009.

[30] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 347–357, 2009.

[31] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," *MICRO*, pp. 115–124, 2010.

[32] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," *MICRO*, pp. 433–444, 2013.

[33] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 222–233, 2013.

[34] S. M. Seyedzadeh, R. Maddah, D. Kline, A. K. Jones, and R. Melhem, "Improving Bit Flip Reduction for Biased and Random Data," *IEEE Transactions on Computers*, Vol. 65, No. 11, pp. 3345–3356, Nov 2016.

[35] J. Daeman and V. Rijmen, "The Block Cipher Rijndael," *Lecture Notes in Computer Science*, Vol. 1820, 2000.

[36] M. Dworkin, "Special Publication 800-38A: Recommendation for block cipher modes of operation," 2001.

[37] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-Efficient Encryption for Non-Volatile Memories," *ASPLOS*, 2015.

[38] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: Smartly EnCRypted Energy EfficienT Non-Volatile Memories," *DAC*, 2016.

[39] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," *DSN*, pp. 333–342, June 2010.

[40] F. Huang, D. Feng, Y. Hua, and W. Zhou, "A wear-leveling-aware counter mode for data encryption in non-volatile memories," *DATE*, 2017.

[41] C. Fruhwirth, "New methods in hard disk encryption," Tech. Rep., Vienna Institute of Technology, 2005.

[42] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices," Tech. Rep. SP 800-38E, NIST, 2010.

[43] K. Minematsu, "Improved Security Analysis of XEX and LRW Modes," *Lecture Notes in Computer Science*, Vol. 4356, 2006.

[44] M. Ball, "Follow-Up on NIST's Consideration of XTS-AES," 2009.

[45] D. Kline, R. Melhem, and A. K. Jones, "Counter Advance for Reliable Encryption in Phase Change Memory," *IEEE Computer Architecture Letters*, Vol. 17, No. 2, No. 2, pp. 209–212, 2018.

[46] S. M. Seyedzadeh, A. K. Jones, and R. G. Melhem, "Enabling Fine-Grain Restricted Coset Coding Through Word-Level Compression for PCM," *CoRR*, Vol. abs/1711.08572, 2017.

[47] S. M. Seyedzadeh, R. Maddah, A. Jones, and R. Melhem, "PRES: Pseudo-random Encoding Scheme to Increase the Bit Flip Reduction in the Memory," *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, (New York, NY, USA), pp. 23:1–23:6, ACM, 2015.

[48] S. M. Seyedzadeh, R. Melhem, and A. Jones, "Improving Sustainability Through Disturbance Crosstalk Mitigation in Deeply Scaled Phase-change Memory," *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, Oct 2018.

[49] J. Li and K. Mohanram, "Write-once-memory-code phase change memory," *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, March 2014.

[50] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, (New York, NY, USA), pp. 190–200, ACM, 2005.

[51] "Overview-SPECCPU2017." `https://www.spec.org/cpu2017/Docs/overview.html`, 2019. Accessed: 2017-03-25.

[52] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 271–282, Feb 2018.

[53] "Benchmark Overview - SPEC Cloud IaaS 2018." `https://www.spec.org/cloud_iaas2018/faqs.html`. Accessed: 2019-11-12.

[54] D. Kline, S. Longofono, S. Ollivier, E. Higgins, R. Melhem, and A. K. Jones, "PREM-Sim: A Resilience Framework for Modeling Traditional and Emerging Memory Reliability," *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 396–409, Oct 2019.

[55] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable Dram Alternative," *SIGARCH Comput. Archit. News*, Vol. 37, No. 3, p. 2–13, June 2009.

[56] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP laboratories*, pp. 22–31, 2009.

[57] S. Wang and E. Ipek, "Reducing data movement energy via online data clustering and encoding," *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 32, IEEE Press, 2016.

[58] J. Zhang, D. Kline, L. Fang, R. Melhem, and A. K. Jones, "RETROFIT: Fault-Aware Wear Leveling," *IEEE Computer Architecture Letters*, Vol. 17, No. 2, pp. 167–170, July 2018.

[59] M. K. Qureshi *et al.*, "Enhancing lifetime and security of phase change memories via Start-Gap wear leveling," *MICRO*, Vol. 14, p. 23, 2009.

[60] J. Yun, S. Lee, and S. Yoo, "Dynamic wear leveling for phase-change memories with endurance variations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 23, No. 9, No. 9, pp. 1604–1615, 2015.

[61] L. Namdeo and H. Nautiyal, "Performance Analysis of Advanced Encryption Standard on FPGA," *International Journal of Computer Applications*, Vol. 153, No. 6, No. 6, 2016.

[62] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An Evaluation of High-Level Mechanistic Core Models," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2014.

[63] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31, No. 7, No. 7, pp. 994–1007, 2012.

[64] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, *Writing Cosets of a Convolutional Code to Increase the Lifetime of Flash Memory.* 50th Annual IEEE Allerton Conference on Communication, Control,and Computing, University of Illinois Urbana-Champaign, 2012.

[65] D. Kline, N. Parshook, , E. Brunvand, R. Melhem, P. K. Chrysanthis, and A. K. Jones, "Holistically evaluating the environmental impacts in modern computing systems," *IGSC*, pp. 1–8, Nov 2016.

[66] C. J. Xue, G. Sun, Y. Zhang, J. J. Yang, Y. Chen, and H. Li, "Emerging non-volatile memories: opportunities and challenges," *CODES+ISSS*, pp. 325–334, IEEE, 2011.

[67] M. L. Gallo and A. Sebastian, "An overview of phase-change memory device physics," *Journal of Physics D: Applied Physics*, Vol. 53, No. 21, p. 213002, mar 2020.