ECS: Error-Correcting Strings for Lifetime Improvements in Nonvolatile Memories

SHIVAM SWAMI, POOVAIAH M. PALANGAPPA, and KARTIK MOHANRAM,

University of Pittsburgh

Emerging nonvolatile memories (NVMs) suffer from low write endurance, resulting in early cell failures (hard errors), which reduce memory lifetime. It was recognized early on that conventional error-correcting codes (ECCs), which are designed for soft errors, are a poor choice for addressing hard errors in NVMs. This led to the evolution of hard error correction schemes like dynamically replicated memory (DRM), error-correcting pointers (ECPs), SAFER, FREE-p, PAYG, and Zombie memory to improve NVM lifetime. Whereas these approaches made significant inroads in addressing hard errors and low memory lifetime in NVMs, overcoming the challenges of underutilization of error-correcting resources and/or implementation overhead (e.g., codec latency, hardware support) remain areas of active research and development.

This article proposes error-correcting strings (ECSs) as a high-utilization, low-latency solution for hard error correction in single-/multi-/triple-level cell (SLC/MLC/TLC) NVMs. At its core, ECS adopts a base-offset approach to store pointers to the failed memory cells; in this work, base is the address of the first failed cell in a memory block and offsets are the distances between successive failed cells in that memory block. Unlike ECP, which uses fixed-length pointers, ECS uses variable-length offsets to point to the failed cells, thereby realizing more pointers to tolerate more hard errors per memory block. Further, this article proposes eXtended-ECS (XECS), a page-level error correction architecture, which employs dynamic on-demand ECS allocation and opportunistic pattern-based data compression to improve NVM lifetime by 2× over ECP-6 for comparable overhead and negligible impact to system performance. Finally, this article demonstrates that ECS is a drop-in replacement for ECP to extend the lifetime of state-of-the-art ECP-based techniques like PAYG and Zombie memory; ECS is also compatible with MLC/TLC NVMs, where it complements drift-induced soft error reduction techniques like ECC and incomplete data mapping to simultaneously extend NVM lifetime.

CCS Concepts: • **Hardware** → *Memory and dense storage*;

Additional Key Words and Phrases: Nonvolatile memories, main memory, reliability

ACM Reference format:

Shivam Swami, Poovaiyah M. Palangappa, and Kartik Mohanram. 2017. ECS: Error-Correcting Strings for Lifetime Improvements in Nonvolatile Memories. *ACM Trans. Archit. Code Optim.* 14, 4, Article 40 (December 2017), 29 pages.

<https://doi.org/10.1145/3151083>

1 INTRODUCTION

The high power consumption and poor potential for technology scaling of DRAM below 22nm (ITR 2011) has spurred research in emerging nonvolatile memories (NVMs) such as phase

Authors' addresses: S. Swami, P. M. Palangappa, and K. Mohanram, Department of Electrical and Computer Engineering, 1238 Benedum Hall, Pittsburgh, PA 15261; email: shs173@pitt.edu, mp.poovaiyah@gmail.com, and kartik.mohanram@gmail.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

© 2017 ACM 1544-3566/2017/12-ART40 \$15.00

<https://doi.org/10.1145/3151083>

change memory (PCM) and resistive RAM (RRAM) (Lee et al. 2009; Zhang and Li 2009b; Qureshi et al. 2009b; Zhou et al. 2009; Burr et al. 2010; Ferreira et al. 2010; Schechter et al. 2010; Qureshi 2011; Xu et al. 2013). However, PCM/RRAM-based NVMs suffer from low endurance due to the small lifetime of 10^{8-10} writes to cell failure (Lee et al. 2009; Zhou et al. 2009; Ferreira et al. 2010; Yue and Zhu 2013). A cell failure manifests as a hard error (i.e., stuck-at-0 or stuck-at-1 fault), and the failed cell cannot be subsequently programmed.

Motivation: Past research (Schroeder et al. 2009; Schroeder and Gibson 2010; Sridharan et al. 2015; Meza et al. 2015b) has shown that memory failures are one of the most common hardware problems that cause machine crashes, and hence pose a serious threat to high system availability in long-term operation. Therefore, if PCM/RRAM is to replace DRAM in high-performance computing systems (that are expected to have 32 to 100 petabytes of main memory (Sridharan et al. 2015)), it is worthwhile to investigate memory error correction techniques that can ensure high memory lifetime and achieve sustained system performance in the presence of high error rates. Additionally, in scenarios where replacing the memory module is costly or infeasible (like web/database servers, space stations, etc.), a high memory lifetime is desirable.

It was recognized early on that conventional error-correcting codes (ECCs) like SEC-DED (Lin and Costello 2004), which are designed for soft errors, are a poor choice for addressing hard errors in NVMs. Since the number of hard errors increases over time, strong ECC architectures are required to ensure complete error recovery. However, strong ECC requires high memory overhead, which increases the storage cost of the NVMs. This led to the evolution of error-correcting pointers (ECPs) (Schechter et al. 2010), which use a constant number of pointer bits for each failed cell on a given memory block. ECP emerged as an attractive alternative to the conventional SEC-DED (72,64) code for improving NVM endurance with equivalent overheads. ECP uses a pointer to point to a failed cell and also stores the correct value for that cell. The standard ECP implementation reserves six ECPs for every 512-bit memory block, enabling each 512-bit block to recover from up to six hard errors (ECP-6 henceforth). In terms of the pages maintained by the operating system, this translates into reserving 384 ECPs per 4KiB page (6 ECPs per 512-bit row $\times$ 64 rows).

However, uniform ECP allocation across the memory leads to underutilization of ECPs, since different rows in a 4KiB page have different error correction requirements. For instance, a weak row (that experiences early cell failures) may require more than six ECPs, which saturates the error-correcting capabilities of the row, resulting in page failure. This page failure is wasteful since the page may have many strong rows (that experience only few early cell failures) and whose error-correcting resources are yet to be exhausted. Prior works (Qureshi 2011; Azevedo et al. 2013) show that very few rows in a page use all the six available ECPs at the time of page failure. Thus, a significant portion of the memory provisioned for ECPs remains unutilized on a failed page.

ECP-based schemes such as Pay-As-You-Go (PAYG) (Qureshi 2011) and Zombie (Azevedo et al. 2013) improve the utilization of ECPs at the cost of added latency (up to one extra read cycle) and hardware overhead. Non-ECP-based schemes like dynamically replicated memory (DRM) (Ipek et al. 2010) and FREE-p (Yoon et al. 2011) require support from the operating system in addition to memory/logic overhead.¹ *Whereas state-of-the-art approaches have made significant inroads in addressing hard errors and low memory lifetime in NVMs, overcoming challenges of underutilization of error-correcting resources and/or implementation overhead (e.g., codec latency, hardware support) remain areas of active research and development.*

¹The role of the OS in deactivating the dead pages is consistent across all the error correction techniques evaluated in this work. However, in addition to deactivating the dead pages, DRM (Ipek et al. 2010) and FREE-p (Yoon et al. 2011) require special support from the OS to manage the pairing of the dead pages and remapping of the dead memory blocks, respectively.

Contributions: This article makes the following three core contributions:

- First, we propose **error-correcting strings (ECS)** as a high-utilization, low-latency solution to recover from NVM hard errors. At its core, ECS adopts a base-offset approach to store pointers to the failed memory cells; in this work, base is the address of the first failed cell in a memory block and offsets are the distances between successive failed cells in that memory block. Unlike ECP, which uses fixed-length pointers, ECS uses variable-length offsets to point to the failed cells, which accommodates more pointers, increasing the ability to tolerate more hard errors per memory block. Whereas variable-length offsets offer a significantly high error correction capability in the ideal scenario, the corresponding codec complexity (logic as well as latency overheads) renders such approaches impractical. To the best of our knowledge, this is the first work that investigates the tradeoff between error correction capability based on variable-length offsets and codec overhead. We propose row-level ECS as a practical solution to realize a high-efficiency variable-length offset error correction solution with a low-latency, low-overhead codec implementation. Row-level ECS uses fixed-length offsets within a row but allows different rows in a page to have variable offset lengths. Note that row-level ECS subsumes fixed-length-offset techniques such as Liu et al. (2014) proposed for hard error correction in flash memories. We demonstrate that the error correction capability of row-level ECS is comparable to the ideal variable-length offset ECS, which assumes no limits on codec complexity.
- Second, we propose **eXtended-ECS (XECS)**, a page-level error correction architecture that extends row-level ECS through dynamic on-demand ECS allocation and opportunistic pattern-based data compression to improve the NVM lifetime for negligible impact to system performance. It is observed that only a few rows in a page experience a large number of errors, and improving the error correction capabilities of these critical rows translates to higher memory lifetime (Qureshi 2011). To this end, XECS increases the error correction capability of the memory by dynamic on-demand allocation of error correction resources to the weaker rows in a page, which also prevents the wastage of error correction resources on stronger rows, resulting in a superior ECS utilization. To further boost the error correction capability, XECS employs opportunistic pattern-based data compression and leverages the recovered free space to store additional ECSs for the rows that have exhausted their ECS resources; thus, XECS improves ECS error correction capability without incurring the memory overhead of storing extra error correction resources. Finally, we perform a comprehensive evaluation to demonstrate that XECS achieves higher NVM lifetime for lower memory overhead in comparison to the state-of-the-art page-level error correction architectures (Schechter et al. 2010) proposed in the literature.
- Third, we show that ECS is a drop-in replacement for ECP to extend the NVM lifetimes of state-of-the-art ECP-based techniques like PAYG (Qureshi 2011) and Zombie memory (Azevedo et al. 2013), without significant modifications to the existing architectures of these techniques. We also demonstrate the integration of XECS with PAYG and Zombie memory for further lifetime improvements. Finally, we show that row-level ECS and XECS complement drift-induced soft error correction techniques (based on ECC) and incomplete data mapping (IDM) (Niu et al. 2013; Seong et al. 2013; Yoon et al. 2013) to simultaneously reduce both hard and soft errors in multi/triple-level cell (MLC/TLC) NVMs.

Evaluation and results: We evaluate the memory lifetime of our proposed architectures—row-level ECS and XECS—using a page-level memory lifetime simulator on memory traces from the SPEC CPU2006 (Henning 2006) benchmark suite. Our results are organized as follows: First, we show that row-level ECS (XECS) improves NVM lifetime by $1.12\times$ ($2\times$) over ECP-6. Second, we

perform a sensitivity analysis of the error correction capabilities of row-level ECS and XECS versus the coefficient of variation (CoV) of cell lifetime. We observe that lifetime improvements (not the lifetime) from row-level ECS (XECS) are better for higher CoV values, reaching up to $1.37\times$ ($3.10\times$) over ECP-6 for a CoV of 0.35. To establish an upper bound on row-level ECS, we also compute the memory lifetime of ideal row-level ECS that allows variable-length offsets within a row. Our results show that ideal row-level ECS improves memory lifetime by $1.51\times$ over ECP-6, which is comparable to the lifetime improvement ($1.37\times$) from practical row-level ECS. Third, we evaluate the impact of XECS on system performance (measured in instructions per cycle (IPC)) using the MARSS (Patel et al. 2011) full-system simulator. Our results show that for early to mid memory lifetime, XECS does not impact IPC; later in memory lifetime, when the number of errors becomes large, IPC reduces by 2.5% over ECP-6. Fourth, we perform a comprehensive comparison of various page-level error correction architectures (Schechter et al. 2010) with XECS to investigate the impact of page-level error correction and pattern-based data compression on NVM lifetime. Further, we also evaluate memory lifetime of ECS integrated with PAYG and Zombie memory. Finally, we perform a qualitative evaluation of ECS-integrated error correction techniques for MLC NVMs in the presence of both hard and soft errors. Our results show that for MLC NVMs, a comprehensive solution that integrates row-level ECS with BCH-1 and IDM offers the best tradeoff between memory lifetime, memory overhead, and soft error rate.

The rest of this article is organized as follows. Section 2 describes ECS theory and row-level ECS. Section 3 explains the XECS architecture. Section 4 describes the evaluation methodology. Section 5 presents results. Section 6 discusses and evaluates ECS for MLC NVMs. Section 7 discusses ECS in the context of NAND flash-based memories. Section 8 discusses related work, and Section 9 concludes the article.

2 ERROR-CORRECTING STRINGS

ECSs are a low-overhead hard-error correction technique for NVMs that improves ECPs. In classical ECP, pointers are used to mark the failed memory cells in a given memory block. Depending on the size (S) of the memory block, the pointer length (n) is determined ($n = \lceil \log_2 S \rceil$). Since an SLC stores 1 bit of logical information, an n -bit pointer is stored using n SLCs. We make a key observation that instead of encoding the absolute addresses of the failed cells, we can encode the offsets (distances) between the successive failed cells and still be able to uniquely point to each failed cell. Encoding the offsets requires fewer bits than encoding the absolute addresses. Whereas this increases the burden of decoding the offsets to recover the absolute locations of the failed memory cells, it can be accomplished by fixing the lengths of the offsets for a memory block and storing this information along with the offsets within ECSs (refer to Section 2.1).

Figure 1 contrasts ECP with ECS. Without loss of generality, consider a 128-bit data block protected by ECP-3 (i.e., ECP architecture with three ECPs per row). Figure 1(a) shows the ECP-3 architecture in which pointers of constant lengths are used for pointing to the failed cells. Along with each pointer is a replacement cell, which stores the correct value for the failed cell given by the pointer. To correct three errors on a 128-bit block, 24 bits ($(7\text{-bit pointer} + 1\text{ replacement bit}) \times 3$) are required by ECP-3. Figure 1(b) shows the ECS architecture that adopts a base-offset approach to record the addresses of the failed memory cells. Here base is the address of the first failed cell in a memory block and offsets are the distances between successive failed cells in that memory block. ECS uses variable-length offsets to point to the failed cells, thereby accommodating more pointers and increasing the ability to tolerate more hard errors per memory block. As shown in the figure, for only 21 bits (17 bits for pointers + 4 replacement bits), ECS is able to correct four errors on the memory block.

Fig. 1. This figure contrasts error-correcting pointers (ECPs) with error-correcting strings (ECSs). (a) ECPs along with the replacement cells pointing to the hard errors. (b) ECSs along with the replacement cells storing the offsets between the errors. ECS adopts a base-offset approach to record the addresses of the failed memory cells, where base is the address of the first failed cell in a memory block and offsets are the distances between successive failed cells in that memory block. For this example, ECP can correct only three hard errors, whereas ECS can correct four hard errors for the same error correction overhead.

Fig. 2. Memory overhead of ECS and ECP by varying the number of pointers available per 512-bit memory block. Unlike ECP, where the size of the error-correcting pointer is constant, the offset size in ECS is variable and decreases as the number of errors in a given memory block increases.

Figure 2 compares the memory overhead of ECP and ECS by varying the number of error-correcting pointers per 512-bit memory block. The memory overhead for ECP can be determined without simulations; however, to determine ECS overhead, we require offset lengths that are dependent on the positions of the failed cells. Hence, to estimate ECS overhead for different failure patterns, we simulate different scenarios using a page-level simulator (explained in detail in Section 4.1). The results reported here are averaged across 1,000 simulations. For each simulation, a 4KiB page is generated by sampling cell lifetimes from a normal distribution (Schechter et al. 2010) and the weakest cells are identified. Due to process variations, there is a significant difference between the lifetimes of the NVM cells even on the same die (Zhang and Li 2009a; Schechter et al. 2010; Qureshi 2011). The coefficient of variation (CoV), defined as σ/μ , is used to express this variability in the cell lifetimes. Here μ is the mean cell lifetime and σ is the standard deviation of the cell lifetimes. Due to the variation in the lifetime distribution, cells with lower lifetimes fail earlier than the others. Our simulator samples the lifetime of each memory cell from a normal distribution for a given mean and CoV. Based on the pattern of cell failures, the base and offset values are computed. It is evident from the results that for a small number of pointers per memory block, the reduction in memory overhead of ECS over ECP is not significant. However, as the number of pointers per memory block increases, the memory overhead of ECS decreases in comparison to the memory overhead of ECP. The explanation is that when a row encounters multiple failures, the distances between successive failures decreases. Consequently, the binary offsets required to encode the distances are correspondingly small. Therefore, for equivalent memory overhead, ECS can accommodate more pointers (i.e., correct more errors than ECP).

Fig. 3. Page organization for row-level ECS: A 4KiB page is organized as 64 rows $\times$ 512 bits/row. Each row is allotted 66 bits for ECSs, which store error-correcting pointers in base-offset form. Offset length is constant within a row, while offsets of different rows have different lengths.

2.1 Row-Level ECS

So far, we have discussed the ideal ECS scenario, where we assume that the memory controller always knows the lengths of the offsets for each row (512-bit memory block in a 4KiB page). However, in practice, this cannot be guaranteed unless each row has offsets of the same length or extra metadata is included to point to the first and the last bit of each offset. To address this problem, we propose row-level ECS as a fixed-length-offset compromise that provides a middle-ground solution to derive the maximum benefits of ECS while also simplifying the process of decoding the offsets for error correction. Row-level ECS allows each 512-bit row in a page to have offsets of different lengths; however for a given row, all offsets are of the same length. The offset length is recorded along with the ECSs for each row, in a fixed-width offset-decoding field of 2 bits (denoted B_1B_0). The encoding 00 = 6-bit offset, 01 = 7-bit offset, 10 = 8-bit offset, and 11 = 9-bit offset is used. The 9-bit offset is equivalent to conventional ECP-6. We limit the memory overhead of row-level ECS to 66 bits for a 512-bit memory block.

2.2 Row-Level ECS: Memory Organization

Figure 3 shows the organization of a 4KiB page in memory with ECSs appended as metadata at the end of each row within the page. As stated earlier, all offsets are of the same length within a row, but different rows may have different offset lengths. Our simulation studies show that the four most frequently occurring offset lengths for a 512-bit row are 6 bits, 7 bits, 8 bits, and 9 bits. Therefore, depending on the location of the errors, one of these four offset lengths is used in the row-level ECS. For example, if all the offsets for a row (including the base, i.e., offset from the 0th bit of that row) are 6 bits or fewer, we append 00 to the ECSs for that row. Upon decoding, 00 informs the error-correcting circuitry that all the offsets (and the base) in the ECSs for this row are 6 bits. With 6-bit offsets, up to nine errors can be corrected by row-level ECS. In addition to nine pointers, we require nine replacement cells in order to hold the correct values for the nine failed cells. Thus, the total number of cells required per row is 9 offset $\times$ 6 bits/offset + 9 replacement cells + 2 offset bits = 65.

To the best of our knowledge, the closest work that uses offset-based coding for error correction is EC-Cache (Liu et al. 2014), which complements low-density parity check (LDPC) codes for error correction in flash memories. However, EC-Cache is a specific case of row-level ECS; that is, EC-Cache restricts all pointers to 8 bits. In contrast, row-level ECS uses pointers of 6, 7, 8, and 9 bits depending on the requirement of a row. Table 1 lists all row-level ECS scenarios.

Table 1. Offset Lengths with Row-Level ECS

Offset Bits ($B_1 B_0$)	Offset Size (in Bits)	Number of Pointers	Total ECS Size (in Bits)
0 0	6	9	$2 + 6 \times 9 + 9 = 65$
0 1	7	8	$2 + 7 \times 8 + 8 = 66$
1 0	8	7	$2 + 8 \times 7 + 7 = 65$
1 1	9	6	$2 + 9 \times 6 + 6 = 62$

Fig. 4. ECS codec: To reduce the computation latency, ECS metadata is decoded in parallel as 6-bit, 7-bit, 8-bit, and 9-bit offsets. Based on the offset-decoding bits B_1 and B_0 , appropriate data is forwarded to the adder to generate the absolute address of the failed cell. Using the absolute address, a 9-to-512 decoder uniquely identifies the faulty bit and replaces it with the correct value.

Note that row-level ECS stores a 512-bit memory block and its corresponding 66-bit ECS in the same memory array. Hence, instead of using a separate chip for ECSs, the capacity of existing chips is increased to accommodate the ECS bits. This approach is consistent with the ECP-based architecture proposed in Schechter et al. (2010).

2.3 Row-Level ECS: Error Correction Mechanism

The error correction codec for row-level ECS is shown in Figure 4. Along the lines of Schroeder et al. (2009) and Yoon et al. (2011), we integrate this codec on the processor-side memory controller and increase the size of the data bus from 64 bits to 72 bits to transfer metadata along with data from the memory module to the processor-side memory controller. Once the metadata reaches the codec, it is decoded based on the offset length (6-bit, 7-bit, 8-bit, and 9-bit). The decoders are simple shift registers that truncate the base and the offsets into 6-, 7-, 8-, or 9-bit strings. To minimize the computation latency, decoding of different offset lengths is done in parallel inside the ECS codec. The decoded offsets are added to generate the absolute addresses of the failed memory cells. Note that we use 9-bit adders to generate the absolute addresses of the failed memory cells. Hence, pointers (base/offsets) present in the ECS are converted to 9-bit binary strings before being forwarded to the adders. All the pointers that are smaller than 9 bits are appended with zeros to generate uniform 9-bit strings. Also, since row-level ECS supports up to nine 6-bit pointers, after conversion

of each pointer to a length of 9 bits, the size of the ECS is 90 bits ($9 \text{ pointers} \times 9 \text{ bits/pointer} + 9 \text{ replacement bits}$). If the number of pointers is less than 9, only a subset of the 90 bits is used for error correction. For example, with six pointers of 9 bits ($B_1B_0 = 11$), only the first 60 bits ($6 \text{ pointers} \times 9 \text{ bits/pointer} + 6 \text{ replacement bits}$) are valid and used for error correction; the last 30 bits are left unused.

Depending on the offset-decoding field B_1B_0 , appropriate metadata is selected by the multiplexer and forwarded to the adders for generating the absolute addresses of the failed memory cells. Using the absolute address, a 9-to-512 decoder selects the faulty bit, which is then XORed with the replacement bit to generate the correct value. During a write, if the stuck-at value (erroneous bit) is not the value that should be written in that cell, the corresponding replacement bit is changed to 1; otherwise, the replacement bit is 0. To determine if error correction is necessary at all (i.e., at least one error), we “OR” all the ECS replacement bits (maximum 9) together. If the result is 0, it implies no error correction is necessary for the row and the data is forwarded to the last-level cache (LLC) as is; otherwise, if the “OR” evaluates to 1, error correction is performed and the error-corrected data is forwarded to the LLC. The latency and area overhead of this codec (evaluated using the Synopsys Design Compiler for 45nm technology) is 3ns and 0.12mm^2 , respectively.

Our design leverages the fact that as the number of errors in a row increases, the offsets between them decrease. Due to a reduction in offset size, it is possible to accommodate more pointers in comparison to conventional ECP-6. Hence, the weaker rows can be allotted up to nine pointers with row-level ECS, but only six pointers with ECP-6.

2.4 Handling Failures in ECS Metadata

A failure in a replacement cell is handled by pointing the next pointer (ECS offset) to the cell whose value was stored in the failed replacement cell. This is accomplished by storing zero in the next pointer and using the replacement cell of the next pointer to store the correct value. The pointer at the higher index takes precedence over the one with the lower index. Note that since replacement bits are interleaved with pointer bits in the metadata, the same bit location can be used as a pointer bit in one row and a replacement bit in a different row.

We explain error correction of an erroneous pointer (EP) using the flowchart in Figure 5. The steps involved are as follows: (1) It is identified if spare ECS bits are available; in the absence of spare ECS bits, EP cannot be corrected and the page is marked as dead. (2) If spare ECS bits are available, EP is compared with its true value (i.e., the value EP is supposed to be storing). (3) If EP is less than its true value (i.e., the offset is smaller than intended), the replacement bit of the EP is set to 0 and the adjacent ECS bits are employed to store the offset between the EP and the true value. (4) If the EP is greater than the true value, it is checked if EP can be pointed to a value smaller than the true value. (5) If the output of step (4) is “yes,” then EP is reset to point to a value that is smaller than the true value, and step (3) is repeated. However, if pointing EP to a smaller value is not possible (i.e., output of step (4) is “no”), the page is marked as dead. Note that since ECS metadata receives very low write traffic, failures in the pointer/offset cells are more likely to occur at the time of manufacture than due to wearout. Such failures can be identified and corrected during post-manufacturing testing (Kannan et al. 2013).

3 EXTENDED-ECS (XECS)

In this section, we describe eXtended ECS (XECS), a page-level error correction architecture that extends row-level ECS through dynamic on-demand ECS allocation and opportunistic pattern-based data compression to improve NVM lifetime for negligible impact to system performance.

Fig. 5. Flowchart for error handling in ECS pointer bits.

3.1 XECS: Dynamic Memory Allocation for Error Correction

Without exception, ECP-6 and row-level ECS uniformly allocate error correction resources across rows in a page. However, in practice, only a few weak rows in a page exhaust all of their error correction resources, whereas a majority of (stronger) rows in a page consume only a part of their error correction resources. Thus, in scenarios where a weak row requires more than its allocated error correction resources, a premature page failure occurs. This page failure is wasteful since the page may have many strong rows (that experience few early cell failures) whose error correction resources are yet to be exhausted.

Such scenarios can be studied in detail by simulating a 4KiB page using the cell lifetime model (Schechter et al. 2010) and identifying the six weakest cells of each row (since ECP-6 can tolerate six cell failures per row), one of which ultimately causes the page to fail. The rows that experience four cell failures or fewer by the time the page fails are the strong rows, while the rows that experience more than four cell failures by the time the page fails are the weak rows. Prior works (Qureshi 2011; Azevedo et al. 2013) as well as our own simulations (refer to Figure 6) show that only a few rows in a page consume all the six available ECPs at the time of a page failure. Thus, a significant portion of the memory provisioned for ECPs remains unutilized on a failed page.

In order to improve the ECS utilization in a 4KiB physical page (64 rows $\times$ 512 SLCs per row), XECS initially allocates only a 48-bit ECS metadata per 512-bit row (as opposed to 66-bit ECS metadata in row-level ECS). XECS also provisions two additional 512-bit XECS rows per page to store the extra ECS pointers for the rows that have exhausted their ECS metadata. Since a majority of rows in a page experience four cell failures or fewer even at the end of the page's lifetime, a 48-bit metadata per row is sufficient to recover from cell failures for most of the rows. For weak rows in a page (i.e., rows that experience more than four cell failures), extra memory cells are allotted in the XECS rows for storing additional ECS pointers. Each XECS row can accommodate ECS pointers for four weak rows; that is, 128 bits of metadata are provisioned per weak row. With 128 extra bits to store pointers, a weak row can recover from at least 12 more cell failures (the largest ECS requires a 9-bit pointer and 1 replacement bit, i.e., 10 bits per cell failure). Thus, by leveraging the ECS resources of strong rows, the XECS architecture allows eight weak rows

Fig. 6. This figure illustrates the cell failure distribution among the rows in a 4KiB physical page (64 rows $\times$ 512 SLCs per row) at the time of page failure with ECP-6. As shown in the figure, there is a significant disparity in the error correction requirements of different rows; only 6% of the rows experience more than four errors until page failure. Due to this, a uniform ECP/ECS allocation scheme is less efficient in comparison to a scheme that dynamically allocates error correction resources according to the pattern of cell failures.

(4 weak rows per XECS row $\times$ 2 XECS rows) to recover from at least 16 cell failures per row (four cell failures are handled by metadata and 12 by XECS row). The memory organization in the XECS architecture is explained in Section 3.3.

3.2 XECS: Leveraging Pattern-Based Data Compression for Error Correction

Data from real-world applications exhibit a significant amount of redundancy (Alameldeen and Wood 2004; Pekhimenko et al. 2012; Palangappa and Mohanram 2016). Data compression schemes exploit these redundancies to store data in a compact form, thereby saving memory. XECS uses a cache line-level pattern-based data compression technique called Base-Delta-Immediate (BAI) (Pekhimenko et al. 2012) to compress the data stored in the main memory and uses the freed space to store additional ECSs, which translates to higher memory lifetime. Note that XECS is agnostic to the choice of compression technique. Our simulations on memory traces from SPEC CPU2006 (Henning 2006) benchmarks show that on average, BAI compresses 46% of write accesses and the maximum size of compressed data in a row is 36 bytes. Hence, at least 28 bytes are unused in a row when the row stores compressed data in BAI form. For a compressed row (i.e., a row storing BAI compressed data), XECS leverages the free space in the compressed row for storing the extra ECS bits to recover from more than four cell failures. This information is recorded in an eXtension flag and used during a read for error correction. Note that when a compressed row becomes uncompressed, its ECS pointers are transferred to the XECS row until the row becomes compressed again (explained in detail in Section 3.4).

3.3 XECS: Memory Organization

Figure 7 shows the XECS memory organization for a 4KiB physical page with metadata for error correction appended at the end of each row. The primary feature of XECS data organization is that it repurposes a part of the metadata of all the rows in a page to obtain two 512-bit XECS rows, which can be used for storing ECS pointers for weak rows in the page. XECS data organization uses four flags to support its dynamic on-demand allocation of error correction resources to the weak rows in a page. First, for a given row, a 1-bit compression flag indicates whether the data stored in that row is in compressed/uncompressed form. Second, a 2-bit eXtension flag per row indicates the location of the ECS pointers for that row. The extra ECS pointers for a weak row are stored either within that row (only if it is a compressed row) or in one of the two XECS rows. The eXtension flag uses the following encoding: 00 = ECS pointers stored in the metadata only (robust

Fig. 7. XECS data organization of a 4KiB physical page. Each row is appended with 48-bit metadata to enable up to four hard error corrections per row. For weak rows (hard errors > 4), two XECS rows are provided to store extra ECS pointers. Note that the XECS rows are obtained by reducing the ECS pointer memory of robust rows and do not incur additional memory overhead; total memory overhead of XECS is 12.79%. Additionally, to boost ECS error correction capability, XECS employs opportunistic pattern-based data compression and leverages the free space recovered from data compression to store additional ECS pointers for the rows that have exhausted their ECS memory. Note that ECS pointers for only uncompressed weak rows are stored in the XECS rows.

row), 01 = ECS pointers stored in the free space of the compressed row, 10 = ECS pointers stored in the first XECS row, and 11 = ECS pointers stored in the second XECS row. Note that each XECS row is partitioned into four 128-bit segments to store ECS pointers for four different rows. Third, a 2-bit eXtension pointer flag points to one of the four segments inside the XECS row that stores the extra ECS pointers for that row. Finally, a 2-bit offset flag records the ECS pointer (base/offset) length for a row, similar to row-level ECS (Section 2.1).

When a weak row (with extra ECS pointers stored in one of the XECS rows) is compressed, XECS transfers the extra ECS pointers (of that weak row) from the XECS row to the free space (recovered from compression) inside that row and marks the corresponding 128-bit segment (which was holding those extra ECS pointers) inside the XECS row as invalid. Each 128-bit segment in an XECS row has a valid bit to indicate whether the ECS pointers stored in the segment are valid or not. The four valid bits (one for each segment) are stored in the metadata appended at the end of an XECS row. The remaining metadata (46-bit metadata - 4 valid bits = 42-bit metadata) is used for storing ECS pointers for error correction within the XECS row.

3.4 XECS: Memory Operations

In this section, we explain the read and write operations for the XECS architecture.

Read: The processor-side memory controller initiates a read by issuing a row access strobe (RAS). This loads the entire physical 4KiB page, along with the metadata for each row and the XECS rows into the row buffer (typically 4 to 8KiB in size (Yoon et al. 2011; Qureshi 2011)). Following this, a column access strobe (CAS) is issued to retrieve the desired cache line from the row buffer over eight clock cycles. Similar to error correction in DRAM-based main memories (Schroeder et al. 2009), we increase the size of the data bus from 64 bits to 72 bits to allow 8 bits of metadata to be transferred per clock cycle. The 48-bit metadata gets transferred to the processor-side memory controller in only six clock cycles; during the remaining two cycles, the metadata is used to initiate error correction by the XECS codec. The eXtension flag indicates the location of ECS pointers for the accessed row. If the extra ECS pointers are stored on one of the XECS rows (eXtension flag = 10/11), a

second read to the appropriate XECS row is issued to retrieve the remaining ECS pointers. However, it is important to note that the read to the XECS rows does not incur a row opening delay, since the entire page is already cached inside the row buffer. Next, based on the offset bits (B_1B_2), the XECS codec performs base-offset address computation, similar to the row-level ECS. Finally, if the compression bit is set, data decompression is performed before forwarding the cache line to the LLC.

Write: A memory write is preceded by a read to implement data comparison write (DCW)—a commonly used technique to improve NVM endurance by eliminating redundant writes to memory (read-modify-write) (Yang et al. 2007; Lee et al. 2009; Zhang and Li 2009b; Qureshi et al. 2009b; Zhou et al. 2009; Burr et al. 2010; Ferreira et al. 2010). The read of the metadata and ECS pointers stored in a compressed row is performed during the read operation of DCW. This information is used to identify the location of ECS pointers and replacement bits for updating them during the write. Next, the data is compressed using BAI compression before it is written to memory. However, if the data is incompressible, it is written as is to memory.

For a weak row (i.e., a row that has experienced more than four cell failures), its extra ECS pointers are stored either in one of the XECS rows (for uncompressed data) or in the free space within that row itself (for compressed data). Note that although XECS leverages the free space in a compressed row for storing extra ECS pointers, it does not significantly increase memory wear of the compressed row. This is because ECS pointers only change when a new cell failure occurs (at most 16 in a row), and hence remain mostly unmodified across writes. When an uncompressed row becomes compressed, its extra ECS pointers are migrated from the XECS row to the freed space within that row. In contrast, when a compressed row becomes uncompressed, its extra ECS pointers are migrated from the free space within that row to one of the XECS rows. The XECS row, which has a segment with the valid bit set to 0, is selected for storing the extra ECS pointers for a weak uncompressed data row; the valid bit for the corresponding segment is set to 1.

As is necessary for all NVMs, a write is followed by a verify-read to determine if any new cell failures have occurred during the write (Ipek et al. 2010; Schechter et al. 2010; Qureshi 2011; Azevedo et al. 2013). If one or more cell failures are encountered on the verify-read, the memory controller rewrites the correct data by invoking the underlying hard error correction scheme (ECS in this case). If error correction is not possible, the page is marked dead and is no longer visible to the operating system.

3.5 XECS Codec

We designed the XECS codec with four levels of carry look-ahead adders, which results in $15(2^4 - 1)$ adders. Each adder is 9 bits wide since the maximum size of an offset is 9 bits. This amounts to a total logic overhead of $\approx 5k$ two-input NAND gates for XECS adders. Furthermore, the latency of an n -bit carry look-ahead adder is $3 + 2\lceil\log_2 n\rceil$ gate delays; hence, the total latency of the XECS adder circuit is $4 \times (3 + 2\lceil\log_2 n\rceil)$ gate delays. The computation of the absolute addresses of the failed cells is followed by error correction, which requires 9-to-512 decoders and XOR gates, similar to the codec shown in Figure 4. A parallel prefix circuit for error correction designed along the lines of Schechter et al. (2010) and Azevedo et al. (2013) amounts to a total gate delay overhead of $O(\log_2 m + \log_2 d)$ for m errors using d -way row decoders. The area and latency for a codec that can handle 16 errors in a row (in the worst case) was evaluated using Synopsys Design Compiler tool to be 0.25mm^2 and 4.2ns , respectively, for 45nm technology. The BAI compression engine (0.5ns latency (Pekhimenko et al. 2012)) is integrated with the XECS codec inside the processor-side memory controller.

3.6 State of the Art in Page-Level Error Correction

We distinguish XECS from the state-of-the-art page-level error correction architecture proposed in Schechter et al. (2010), which also employs a spare row per page to improve the error correction capability. Unlike XECS, which employs variable-length ECS pointers, Schechter et al. (2010) employs two types of fixed-length ECPs—10-bit row-level ECPs (9-bit pointer + 1 replacement bit), which are stored alongside the rows, and 16-bit page-level ECPs (15-bit pointer + 1 replacement bit), which are stored in the spare row assigned to every page. Row-level ECPs are exclusively used for error correction within their corresponding rows; in contrast, page-level ECPs are used for augmenting the error correction of a weak row (in that page) that has exhausted all its row-level ECPs. We perform a comprehensive evaluation of various ECP-/ECS-based page-level error correction architectures (in the presence/absence of pattern-based data compression) and compare them with XECS in Section 5.4.

4 EVALUATION METHODOLOGY

Our evaluation of ECS/XECS is based on (1) trace-based lifetime evaluation using a page-level memory lifetime simulator and (2) full-system simulations to evaluate the system-level performance of our proposed ECS architectures.

4.1 Lifetime Evaluation

A memory's lifetime is measured in terms of the number of writes to the memory before its capacity reaches zero. Memory capacity is measured in units of pages, where each physical page can map 4KiB of logical data and 4KiB is the size of a page in the operating system. Following past research (Ipek et al. 2010; Schechter et al. 2010; Yoon et al. 2011; Qureshi 2011), we assume that a 4KiB logical page is mapped to a physical page of 4KiB, organized as 64 rows of 512 bits. When a physical page encounters an uncorrectable hard error, it is mapped out by the operating system and marked as dead. As the number of dead pages increases, memory capacity is reduced, reaching zero at the end of memory lifetime. However, in practice, effective memory lifetime is measured until the memory capacity reaches 90% (Yoon et al. 2011; Niu et al. 2013); this work follows this convention.

Along the lines of Schechter et al. (2010), Yoon et al. (2011), Qureshi (2011), and Azevedo et al. (2013), we sample the lifetime of each memory cell from a normal distribution with mean = 10^8 writes and CoV = 0.25. The write-back unit is 64 bytes, and for a given write, a cell changes value with a bit-flip probability (BFP) computed from SPEC CPU2006 (Henning 2006) memory traces in the presence of horizontal and vertical Start-Gap wear leveling (Qureshi et al. 2009a; Young et al. 2015). For XECS, the simulator performs BAI compression prior to wear leveling. Whereas data compression increases the BFP of certain cells, it also reduces the number of cells programmed on a write. This nonuniform memory wear is countered by wear leveling, which evenly distributes writes across memory cells (both within and across rows), resulting in uniform memory wear (i.e., BFP = modified bits/total bits in a cache line). Table 2 lists the BFP values used in our simulations. Additionally, following Yoon et al. (2011), to compute memory lifetime in units of years, we use a constant write-back rate of 66.67 million writes per second (i.e., one-third peak data rate of a 12.8GiB/s DDR3-like channel).

As the simulation proceeds, the number of writes for each cell reaches its lifetime, causing the cell to fail.² On encountering a cell failure, the simulator repairs the page according to the

²Since the maximum number of cell failures that can be recorded by row-level ECS (XECS) is nine (16) per row, offsets of a row are written at most nine (16) times in a page's lifetime. These offset rearranging writes are insignificant in comparison to the millions of writes seen by a row in a page's lifetime; hence, we do not consider them in our simulations. However,

Table 2. Bit-Flip Probability (BFP) for SPEC CPU2006 Benchmarks

Benchmark	BFP (Uncompressed)	BFP (XECS)
perlbench	0.24	0.24
soplex	0.34	0.29
astar	0.39	0.28
hmmer	0.22	0.11
sjeng	0.40	0.22
h264ref	0.19	0.12
namd	0.31	0.21
bwaves	0.18	0.14
milc	0.49	0.28
zeusmp	0.43	0.23
povray	0.14	0.13
calculix	0.33	0.21
lbm	0.37	0.3
leslie3d	0.30	0.16
gemsFDTD	0.43	0.24
Geometric mean	0.30	0.20

In order to capture the effect of added writes by XECS when a cache line switches from a compressed to uncompressed state, we compute net occurrence of such switches on a benchmark run. On every switch, we increase the number of programmed cells on that write by 128, since the extra ECS bits can occupy up to 128 bits on an XECS row (refer to Section 3.3). Hence, BFP for XECS also includes the extra writes that are incurred when a compressed cache line becomes uncompressed.

Table 3. Full-System Configuration

Parameter	Attributes
Core	4 out-of-order x86 cores
L1 (instruction/data) cache	128KiB, 8-way, private, 2ns latency
L2 cache	2MB, 8-way, shared, 5ns latency
NVRAM	16GiB, 75ns (read), 150ns (write) (Choi et al. 2012)

implemented error correction scheme. If error correction is not possible, the page is marked dead and the wear rate of the remaining pages is increased. This method is consistent with the one explained in Schechter et al. (2010).

4.2 Performance Evaluation

We use the MARSSx86 (Patel et al. 2011) full-system simulator to evaluate the impact of error correction on system performance. MARSS was configured to simulate a standard four-core out-of-order system running at 3GHz. We model write-back caches with a cache line size of 64 bytes; a write request arrives at the main memory only on eviction from the last-level cache. The simulation configuration is presented in Table 3.

we fully account for the writes seen by the ECS replacement bits, since they are written much more frequently than the ECS offset bits. Note that the initial wear rate of an unused replacement bit remains 0 until it is deployed to replace a failed cell. Further, writes to the ECS bits are automatically wear leveled due to wear leveling of NVM data cells.

Table 4. Composite Workloads Using SPEC CPU2006 Benchmarks, for Full-system Evaluation

Workloads:	WD ₁	WD ₂	WD ₃	WD ₄	WD ₅	WD ₆	WD ₇	WD ₈	WD ₉
Constituent benchmarks:	perlbench soplex bwaves lbm	perlbench soplex milc povray	h264ref astar milc lbm	h264ref sjeng bwaves povray	perlbench leslie3d GemsFDTD lbm	bzip2 gcc sphinx3 xalancbmk	bzip2 gcc mcf omnetpp	bzip 2 mcf namd omnetpp	GemsFDTD gcc omnetpp xalancbmk

Workloads: To model the real-world usage, we use nine composite workloads with each workload containing four benchmarks from the SPEC CPU2006 (Henning 2006). These benchmarks reflect a variety of integer- and floating-point-based applications used by modern computing systems. During the full-system simulation, each benchmark in a workload is tied separately to a core of our simulated machine. This approach is consistent with Palangappa and Mohanram (2016). Table 4 lists the workloads along with their constituent benchmarks.

5 RESULTS

This section presents the lifetime and IPC evaluations for the error correction techniques proposed in this work. We use ECP-6 (Schechter et al. 2010), which recovers from six hard errors per 512-bit row, as our baseline. The results are organized as follows. Section 5.1 presents the trace-based lifetime results for row-level ECS and XECS. Section 5.2 evaluates the sensitivity of row-level ECS and XECS to CoV. Section 5.3 presents IPC results to evaluate the impact of error correction on system performance. Section 5.4 presents a comprehensive comparison of various page-level error correction architectures (in the presence/absence of pattern-based data compression) with XECS. Section 5.5 shows that ECS is a drop-in replacement for ECP to extend the lifetime of state-of-the-art ECP-based techniques like PAYG (Qureshi 2011) and Zombie memory (Azevedo et al. 2013); note that the PAYG + XECS and Zombie + XECS architectures are also evaluated. Finally, Section 6 demonstrates that row-level ECS and XECS complement drift-induced soft error correction techniques to simultaneously reduce both hard and soft errors in multi/triple-level cell (MLC/TLC) NVMs.

Summary: The two key differences between the proposed architectures—row-level ECS and XECS—are as follows. First, the error correction capability of row-level ECS is up to nine errors per 512-bit row, whereas XECS can recover from up to 16 errors in a 512-bit row. This results in a $1.78\times$ higher memory lifetime of XECS in comparison to row-level ECS. Second, due to its lower error correction capability, row-level ECS employs a simpler codec that minimally impacts performance. In contrast, XECS requires a relatively complex codec along with a compression engine, resulting in a 2.5% performance loss in the presence of a high error rate. Thus, XECS trades off performance for memory lifetime. We summarize our results (for CoV = 0.25) along with other state-of-the-art error correction techniques in Table 5.

5.1 NVM Lifetime

Figure 8 compares the memory lifetime of XECS with ECP-6 and its derivatives like PAYG (Qureshi 2011) and Zombie memory (Azevedo et al. 2013), and non-ECP-based hard error correction techniques like SAFER (Seong et al. 2010b) and FREE-p (Yoon et al. 2011). We observe that XECS improves memory lifetime by $2\times$ over ECP-6, and the memory lifetime of XECS is also higher than any ECP- and non-ECP-based error correction technique. The reason for this significant lifetime improvement is that XECS performs dynamic on-demand allocation of error correction resources to the weak rows in a page, as opposed to ECP-6/row-level ECS, which statically reserves memory to store ECPs/ECSs on a per-row basis. XECS ensures that within a page, weaker rows are assigned more pointers and the stronger rows do not consume unnecessary pointers.

Table 5. This Table Compares Error Correction Techniques (for CoV = 0.25) in Terms of Memory Overhead, Memory Lifetime, and IPC

Technique	Memory Overhead (%)	Lifetime (Norm. ECP-6)	IPC [†] (Norm. ECP-6)
ECP-6	12.00	1.00×	1.00
PAYG (Qureshi 2011)	4.13	1.18×	0.85
Zombie (Azevedo et al. 2013)	12.50	1.35×	0.93
Row-level ECS	12.89	1.12×	0.99
XECS	12.79	2.00×	0.975

[†]Note that IPC reduction occurs only later in the memory lifetime when the number of errors becomes high. For early to mid lifetime, all the techniques incur negligible performance loss.

Fig. 8. Memory lifetime (in years) with different error correction techniques for mean cell lifetime of 10^8 writes and CoV of 0.25. The lifetime evaluation is done using a page-level memory simulator on memory traces from the SPEC CPU2006 benchmark suite. XECS increases memory lifetime by 100% in comparison to ECP-6. The reason for a high memory lifetime of XECS is its dynamic on-demand ECS allocation policy and its ability to leverage the free space available in the compressed rows to store extra ECS pointers.

Additionally, XECS employs opportunistic pattern-based data compression and leverages the free space in the compressed data rows to store extra ECSs. Benchmarks with high compression ratios such as hmmer, h264ref, and leslie3d result in more than 2× lifetime improvement over ECP-6.

Row-level ECS, which is a direct replacement for ECP-6, improves memory lifetime by 1.12× over ECP-6. This is equivalent to the memory lifetime achieved by ECP-9 (Schechter et al. 2010), which reserves 91-bit metadata to recover from nine cell failures. Note that row-level ECS cannot outperform ECP-9 under every scenario; in the scenarios where the largest offset (i.e., number of operational memory cells between two stuck-at faults) is greater than 256 and is concurrent with multiple smaller offsets (cluster/burst failures), the error correction capability of ECP-9 is better than ECS. This is because in such scenarios, the size of ECS pointers is dictated by the largest offset, which is 9 bits ($\lceil \log_2 \text{largest offset} \rceil$). As a result, the error correction capability of ECS is reduced to only six failures (since we provision 66-bit ECS metadata and each cell failure requires a 9-bit pointer and 1 replacement bit). In contrast, ECP-9 can still recover from nine failures because it provisions 91-bit ECP metadata (i.e., 38% extra memory overhead in comparison to row-level ECS). However, since our NVM lifetime model—built along the lines of Schechter et al. (2010), Yoon et al. (2011), Qureshi (2011), and Azevedo et al. (2013)—assumes that cell failures are independent and identically distributed, such cluster/burst failures are not a dominant scenario in our simulations. Further, since cluster/burst failures usually occur due to layout-dependent variations (that lead to spatial correlation in memory lifetimes of nearby cells), these failures can be better handled using variation-aware

Fig. 9. Impact of CoV on lifetime improvements from various error correction techniques. We observe that XECS becomes more effective with high CoV; a high CoV results in more early cell failures, reducing the offset between the failures, and thereby enabling more pointers to be stored in a given memory block.

memory designs (e.g., PV-aware PRAM (Zhang and Li 2009a)), which are orthogonal to this work.

5.2 Sensitivity to CoV

In this section, we evaluate memory lifetime with different CoV values (0.15 to 0.35) (Schechter et al. 2010; Seong et al. 2010b; Yoon et al. 2011; Niu et al. 2013). Varying the CoV is an easy and effective means to model different cell failure rates resulting due to nonuniform memory accesses and/or imperfect wear leveling. Figure 9 shows the impact of CoV on lifetime improvements from various error correction techniques. Whereas row-level ECS assigns fixed-length offsets within a row (i.e., all offsets in a row are of the same length) and variable-length offsets between the rows on a page, ideal row-level ECS assigns variable-length offsets both within a row and between the rows on a page. As shown in the figure, the error correction capability of row-level ECS is higher in comparison to ECP-6 and comparable to ideal row-level ECS. Since row-level ECS requires marginal overhead (12% for ECP-6 vs. 12.89% for row-level ECS), row-level ECS is an inexpensive replacement for ECP-6.

Furthermore, we observe that the lifetime improvements (not the absolute NVM lifetime) from row-level ECS and XECS are higher for high CoV values, reaching up to 1.37 $\times$ and 3 $\times$, respectively, for a CoV of 0.35. This is because for small CoV values, there are only a few early cell failures, which can be easily corrected using ECP-6. The majority of the cell failures for a small CoV occur around the mean cell lifetime, resulting in an avalanche of cell failures, which is beyond the error-correcting capacities of both ECP and ECS architectures, that is, row-level ECS and XECS. In contrast, a high CoV results in a higher number of early cell failures, which reduces the offset between the failures, thereby enabling more pointers to be stored in the memory reserved for row-level ECS and XECS.

Note that lifetime improvements from PAYG and Zombie memory are also higher for high CoV values, reaching up to 1.42 $\times$ and 2.5 $\times$, respectively, for CoV of 0.35. However, since a high COV results in a large number of early cell failures, PAYG and Zombie memory start incurring high performance overhead (one or two extra read cycles for every read/write) for correcting increased errors during their initial years of operation, making them infeasible in practice. In contrast, since XECS minimally impacts performance (2.5% for XECS vs. 15% (7%) for PAYG (Zombie) memory, discussed in detail in Section 5.3) even in the presence of high error rates, XECS is still preferred

Fig. 10. (a) Latency comparison of XECS with state-of-the-art error correction techniques, over different phases of memory lifetime. [†] Since codec latency is negligible in comparison to memory access latency, the overall latency of XECS is only 1.36× the latency of ECP-6. (b) Impact of error correction latency on IPC for the later part of memory lifetime. IPC reduction (2.5%) for XECS is due to the latency of the codec. For PAYG (Zombie), latency overhead to fetch ECPs degrades IPC by 15% (7%). Note that since the average XECS lifetime (for a moderate CoV of 0.25) is ≈ 13 years (as shown in Figure 9), “late in memory lifetime” refers to the period after the first 9 years of operation. Similarly, for a CoV of 0.35, the average XECS lifetime is 7 years, so the period after the first 5 years of operation constitutes the later part of memory lifetime.

for deployment in memories that experience a large number of errors as well as in scenarios where the manufacturing process is still immature.

5.3 Impact of Error Correction on System Performance

In this section, we evaluate XECS latency and its impact on system performance. Since the codec latency of ECP-6 is negligible, we use ECP-6 as our baseline. Note that row-level ECS is an extension of ECP-6 and incurs insignificant latency (refer to Section 2.3). Figure 10(a) reports total memory latency for different error correction techniques over different phases of memory lifetime. We observe that in the early to mid lifetime phase, all error correction techniques incur memory latency comparable to ECP-6; however, late in the lifetime, when the number of errors is large, the latency overhead of error correction increases only slightly for XECS, but significantly for PAYG and Zombie memory.

The two sources that contribute to the latency overhead of XECS are that (1) XECS requires a four-level adder circuit for error correction (refer to Section 3.5) and (2) uncompressed weak data rows require a second read access to an XECS row. However, it is important to note that the second read to fetch the XECS row does not incur a row opening delay, since the entire page is already cached in the row buffer (4KiB in this work). For this reason, reading the XECS row only requires a CAS and eight clock cycles (data burst); that is, no RAS is required, which reduces the read latency of XECS by $\approx 66\%$ (using DDR3 parameters from Yoon et al. (2011)). In contrast, the latency overhead of error correction becomes significant for PAYG and Zombie memory late in memory lifetime because PAYG and Zombie memory store ECPs for different rows in a page at different memory locations. For this reason, a separate, full-latency read is required to access those ECPs, which increases the total error correction latency of a weak row by two (one) extra read cycles for PAYG (Zombie memory).

We evaluate the impact of error correction on system performance using the MARSS (Patel et al. 2011) full-system simulator. To evaluate the performance in the presence of failed cells, we inject memory errors using a cell failure rate derived from the lifetime simulator, assuming no new failures occur during program execution. This method is consistent with the approach explained in Yoon et al. (2011). Figure 10(b) reports the impact of error correction on system performance. Our simulations show that for the majority of memory lifetime, XECS and other error correction techniques do not impact system performance. However, later in the memory lifetime, due to a 2 to 3× increase in the error correction latency of a weak row (as explained

earlier), the performance loss from PAYG and Zombie memory becomes significant; IPC reduces by 15% (7%) for PAYG (Zombie memory).

Note that for a moderate CoV (i.e., low cell failure rate), these weak rows become dominant only in the later part of memory lifetime; however, for memories that experience a large number of early cell failures as well as in scenarios where the manufacturing process is still immature, such weak rows become dominant early on in memory lifetime. Hence, for such scenarios, PAYG and Zombie memory can impact system performance even in the initial years of deployment. Additionally, in scenarios where replacing the memory module is costly or infeasible (like web/database servers, space stations, etc.), PAYG and Zombie memory cannot be deployed in practice because these techniques cannot ensure sustained performance after initial years of operation. In contrast, since XECS maintains high system performance (in excess of 97.5%) till the end of memory lifetime, it is a more practical error correction technique.

5.4 XECS Versus Page-Level Error Correction and Pattern-Based Data Compression

In this section, we present a comprehensive evaluation of various ECP-/ECS-based page-level error correction and/or pattern-based data compression architectures and compare them with XECS. We simulate the following architectures:

- (1) **Page-level ECP + ECP-6 (PECP)**: This architecture integrates page-level error correction with conventional ECP-6 architecture (Schechter et al. 2010). In this architecture, each row in a 4KiB page is assigned six row-level ECPs and each page is assigned a spare row for storing page-level ECPs. A 1-bit flag is reserved for each row to indicate if the page-level error correction is required for that row. If this flag is set, reading/writing to that row requires reading/writing of the page-level ECPs in the spare row.
- (2) **Compression + ECP-6 (CECP)**: This architecture integrates pattern-based data compression with conventional ECP-6 architecture. Data compression reduces memory wear by reducing the number of updated cells on a write. A 1-bit compression flag indicates whether the data stored in a row is compressed or not.
- (3) **Page-level ECP + compression + ECP-6 (PCECP)**: This architecture integrates pattern-based data compression with PECP architecture. PCECP benefits from both reduced wear (due to pattern-based data compression) and improved error correction capability (due to page-level error correction).
- (4) **Page-level ECS + row-level ECS (PRECS)**: This architecture is similar to the PECP architecture, with ECPs replaced by ECSs. We use the row-level ECS (described in Section 2.1) as the underlying ECS architecture and provision a spare row per page for storing page-level ECS.
- (5) **Compression + row-level ECS (CRECS)**: This architecture integrates pattern-based data compression with row-level ECS architecture (described in Section 2.1).
- (6) **Page-level ECS + compression + row-level ECS (PCRECS)**: This architecture integrates pattern-based data compression with the PRECS architecture.

Figure 11 reports NVM lifetime of the error correction architectures described previously. For brevity, Figure 11 displays only a subset of memory traces from the SPEC CPU2006 (Henning 2006) benchmark suite; however, the geometric mean is computed for all the traces shown in Figure 8. As shown in Figure 11, for the benchmarks with a small compression ratio (i.e., original data width divided by compressed data width) such as perlbench, soplex, and povray, NVM lifetimes of PECP and PRECS are higher than NVM lifetimes of CECP and CRECS, respectively. This is because in the absence of compression, wear rates of PECP (PRECS) and CECP (CRECS) are almost equal, but the error correction capability of PECP (PRECS) is better than CECP (CRECS) due to extra

Fig. 11. NVM lifetimes of various row-level and page-level error correction architectures with/without pattern-based data compression, evaluated using the SPEC CPU2006 benchmarks. Whereas page-level error correction improves NVM lifetime by increasing the error correction capability of the underlying error correction technique, pattern-based data compression improves NVM lifetime by reducing memory wear. Although naively integrating these two techniques further improves memory lifetime, it results in large wastage of error correction resources.

page-level error correction. In contrast, for the benchmarks that have a high compression ratio such as zeusmp and leslie3d, NVM lifetimes of CECP (CRECS) are higher than PECP (PRECS) due to reduced wear rate. Furthermore, we observe that when compression is integrated with page-level error correction architectures (e.g., PCECP and PCRECS), there is a 71% and 93% lifetime improvement, respectively, in comparison to ECP-6 lifetime. The two factors that contribute to this significant improvement are (1) reduced memory wear due to compression and (2) increased error correction capability due to the page-level error correction, which reserves a spare row for storing extra ECPs/ECSs of weak uncompressed rows.

A summary of the results is provided in Table 6. We observe that although PCECP and PCRECS architectures achieve high NVM lifetimes, the memory overhead of such architectures is high due to the problem of underutilization of error correction resources in the underlying ECP/ECS architectures. In contrast, XECS follows a dynamic on-demand resource allocation approach, which improves the utilization of error correction resources, resulting in higher NVM lifetime.

5.5 Integrating ECS/XECS with PAYG and Zombie Memory

PAYG (Qureshi 2011) and Zombie memory (Azevedo et al. 2013) are ECP-based error-correcting techniques that improve memory lifetime while ensuring low memory overhead. Although PAYG and Zombie memory register strong gains over ECP (as shown in Figure 8), it is also possible to replace ECP with ECS in PAYG and Zombie memory to evaluate the impact on memory lifetime. To integrate ECS with PAYG, we use PAYG’s local ECP (per row) as the base for the ECS and store the offsets in PAYG’s global error correction (GEC) pool. Similar to row-level ECS, we use four types of offsets—6-bit, 7-bit, 8-bit, and 9-bit—and store the offset decoding information along with the base pointer. Similarly, for replacing ECPs with ECSs in Zombie memory architecture, we use the row-level ECS organization (explained in Section 2.1) to store the ECSs in the metadata. Once the metadata exhausts all its pointers, the same row-level ECS organization is followed for storing the ECSs in a spare block (on a dead page). Furthermore, we integrate pattern-based data compression with PAYG + ECS (Zombie + ECS) to reduce memory wear and further extend its error correction capability. We refer to PAYG + ECS + compression (Zombie + ECS + compression) as

Table 6. Summary of NVM Lifetime and Memory Overhead of Various Row-Level and Page-Level Error Correction Architectures With/Without Pattern-Based Data Compression

Technique	Lifetime (norm. ECP-6)	Memory overhead
ECP-6	1.00	12.00%
Page-level ECP + ECP-6 (PECP)	1.20	13.67%
Compression + ECP-6 (CECP)	1.40	13.67%
Page-level ECP + compression + ECP-6 (PCECP)	1.71	13.67%
Row-level ECS (RECS)	1.15	12.89%
Page-level ECS + row-level ECS (PRECS)	1.34	14.65%
Compression + row-level ECS (CRECS)	1.61	14.65%
Page-level ECS + compression + row-level ECS (PCRECS)	1.93	14.65%
XECS	1.99	12.79%

Note that although integrating pattern-based data compression with page-level error correction (PCECP/PCRECS) significantly improves NVM lifetime and the results are comparable to XECS, high memory overhead of these architectures makes them less attractive in practice.

Fig. 12. NVM lifetime for PAYG and Zombie using ECP, ECS, and XECS as the underlying error correction mechanisms. Since ECS accommodates more pointers in comparison to ECP, PAYG + ECS (Zombie + ECS) has higher error correction capability than conventional ECP-based PAYG (Zombie). This is reflected in the 6% (5%) lifetime improvement of PAYG + ECS (Zombie + ECS) over conventional PAYG (Zombie). Since PAYG + XECS and Zombie + XECS integrate pattern-based data compression with their corresponding ECS-based architectures, they register even higher lifetime gains (51% for PAYG + XECS; 55% for Zombie + XECS) over PAYG/Zombie memory.

PAYG + XECS (Zombie + XECS). Note that ECS-/XECS-based PAYG/Zombie architectures require the ECS codec (explained in Section 3.5) in place of the conventional ECP codec.

Figure 12 reports memory lifetime of ECP-based and ECS-based PAYG and Zombie memory architectures. Since ECS can accommodate more pointers for a given memory overhead, more error correction is possible with PAYG + ECS (Zombie + ECS) in comparison to conventional ECP-based PAYG (Zombie memory) architecture. For no additional overhead, PAYG + ECS (Zombie + ECS) improves NVM lifetime by 6% (5%) over conventional ECP-based PAYG (Zombie). Furthermore, we observe that NVM lifetime of PAYG + XECS (Zombie + XECS) is 51% (55%) higher than conventional ECP-based PAYG (Zombie memory). However, note that high NVM lifetimes of PAYG + XECS and Zombie + XECS are at the cost of high memory access latency inherent to PAYG and Zombie architectures, which makes these techniques infeasible in practice for performance-critical systems.

6 ECS FOR MLC/TLC NVMS

In addition to hard errors, multi-/triple-level cell (MLC/TLC) NVMs are susceptible to soft errors caused by different types of transient faults. Typically, memory architects employ separate error correction/mitigation techniques to counter hard and soft errors due to the completely different sources of these errors. Whereas hard error correction techniques are agnostic to the underlying NVM technology, soft error correction techniques are customized for the underlying NVM technology (Yoon et al. 2011; Xue et al. 2011; Swami and Mohanram 2017; Mittal 2017). In this section, we integrate row-level ECS and XECS with state-of-the-art drift-induced soft error mitigation and correction solutions in order to synergistically reduce both hard errors and drift-induced soft errors (the major type of soft error in MLC/TLC PCM (Awasthi et al. 2012; Yoon et al. 2013; Seong et al. 2013; Swami and Mohanram 2017)), without incurring high memory overhead. Without loss of generality, this work evaluates ECS with MLC NVMs; the extension to TLC NVMs is omitted for brevity.

Awasthi et al. (2012) proposed using ECC along with memory scrubbing to detect and correct resistance drift errors. Besides write energy and latency, repeated scrubbing of memory adversely affects the endurance of the memory cells. To reduce the scrubbing frequency, Yoon et al. (2013) and Seong et al. (2013) suggest using an MLC as a three-level cell (3LC) by removing the drift-prone intermediate resistance level to delay the refresh interval. This is also known as incomplete data mapping (IDM) (Jiang et al. 2012; Yoon et al. 2013; Seong et al. 2013; Niu et al. 2013; Swami and Mohanram 2016), wherein two physical MLCs together store 3 logical bits.³ MLC NVMs with IDM and a simple ECC (like BCH-1, which corrects a single error in a 64B block) require a refresh interval of several years. To recover from hard errors, Yoon et al. (2013) proposed the mark-and-spare technique, which is a low-overhead ECP implementation. However, mark-and-spare can be used only for a cell stuck at the reset state. For a cell stuck at the set state, the cell is first forced to the reset state by applying a high reverse current. Since forcing a stuck-at-set cell to the reset state is not always possible, mark-and-spare does not guarantee hard error correction under all scenarios.

We integrate and evaluate ECS—a more reliable technique for hard error recovery—in MLC NVMs. First, we integrate row-level ECS from Section 2.1 with IDM-based MLC (i.e., 3LC) NVMs. Since in 3LC mode two physical cells together store 3 logical bits, a 512-bit cache line is stored in 342 physical cells. For this architecture, row-level ECS with eight 7-bit offsets requires 2 bits to store the offset type, 8×7 bits to store the offsets (pointers), and 12 replacement bits (1.5 bits/replacement cell $\times$ 8 cells), a total of 70 bits. Additionally, we employ BCH-1 (10 logical bits per error) for soft error correction. In IDM mode, 80 logical bits (70 logical bits for row-level ECS and 10 logical bits for BCH-1) are stored using 54 MLCs ($70/1.5$), for 15.78% ($54/342$) memory overhead. We also integrate XECS with IDM-based MLC NVMs and employ BCH-1 for soft error correction per row. The total memory overhead of this architecture is 19%.

Table 7 gives a qualitative comparison of various hard+soft error correction techniques for MLC NVMs. We consider a conventional MLC NVM protected with ECP-3 as our baseline. A conventional MLC NVM maps a 512-bit cache line to a row of 256 MLCs; ECP-3 recovers from up to three hard errors per row. Additionally, since an MLC is highly susceptible to resistance drift, BCH-10 code is required to avoid frequent (every 10 seconds) memory refresh. We follow Yoon et al. (2013) to compute the memory refresh period of various MLC PCM architectures. Our results show that despite using a high overhead ECC (BCH-10 = 100 bits or 50 MLCs), a refresh interval of 17 minutes is required for the baseline. This has a high performance penalty and high energy over-

³An MLC consists of four resistance levels, that is, 2 bits/cell. A 3LC is an MLC that operates in only three resistance levels such that two cells together store 3 bits (Jiang et al. 2012; Yoon et al. 2013; Seong et al. 2013). A TLC consists of eight resistance levels, that is, 3 bits/cell.

Table 7. Qualitative Comparison of Soft+Hard Error Correction Techniques for MLC PCM

Storage Mechanism	Hard Error Correction	Soft Error Correction	Refresh Period	Memory Overhead (%)	Lifetime Improvement over MLC with ECP-3
Conventional MLC (2 bits/cell)	ECP-3	BCH-10	17 minutes	25.4	1×
IDM (3LC (Yoon et al. 2013; Seong et al. 2013)) (3 bits/2 cells)	ECP-6	BCH-1	>68 years	13.1	1.26×
IDM (3LC (Yoon et al. 2013; Seong et al. 2013)) (3 bits/2 cells)	Mark-and-spare (Yoon et al. 2013) (up to 6 hard errors)	BCH-1	>68 years	4.9	1.36×
IDM (3LC (Yoon et al. 2013; Seong et al. 2013)) (3 bits/2 cells)	Row-level ECS	BCH-1	>68 years	15.7	1.51×
IDM (3LC (Yoon et al. 2013; Seong et al. 2013)) (3 bits/2 cells)	XECS	BCH-1	>68 years	19.0	2.08×

Refresh period is the time interval between consecutive memory refresh to recover from drift-induced soft errors.

head (Yoon et al. 2013). In contrast, using IDM-based MLC NVM with a simple ECC (BCH-1 = 10 bits or 5 MLCs) significantly reduces the frequency of the refresh operation to greater than once in 68 years.

The last column of Table 7 reports the lifetime improvements for different MLC NVM hard error correction techniques over the baseline. The lifetime of IDM-based MLC NVM with mark-and-spare is only slightly less than the lifetime of IDM-based MLC NVM with ECP-6 because we have assumed (optimistically, to favor mark-and-spare) that up to 90% of the stuck-at-set cells can be forced to the reset state. If a page contains a stuck-at-set cell that cannot be forced to the reset state, mark-and-spare fails to recover from the error and the page is marked as dead. It is clear that IDM-based MLC NVM with row-level ECS for hard error correction and BCH-1 for soft error correction offers the best tradeoff between memory overhead, refresh interval, and NVM lifetime. Note that although XECS can be employed to further enhance NVM lifetime, its memory overhead and performance impact make it less attractive.

7 EXTENSION TO NAND-FLASH-BASED SOLID-STATE DRIVES

ECS/XECS has potential applications for error recovery in MLC/TLC NAND-flash-based solid-state drives (SSDs), which are widely used as secondary storage devices in modern computing systems. First, certain memory blocks in NAND-flash-based SSDs fail early due to manufacturing issues like process variation. In practice, these original bad blocks (OBBs) are replaced with reserved blocks that are provided at the time of manufacture. Discarding these OBBs reduces the pool of reserved blocks early on, which reduces the error correction capabilities of SSDs later in lifetime. In this scenario, XECS can be used to potentially enable reliable operation of these OBBs instead of replacing them with the reserved blocks, thereby saving the reserved-block pool for runtime errors.

Second, NAND-flash-based SSDs are vulnerable to transient errors like data retention and disturbance failures (Cai et al. 2015a, 2015b). Although these errors can be recovered by erasing the erroneous block, advanced ECC schemes like LDPC codes are employed to support error-free reads before the next erase cycle. However, LDPC requires iterative and probabilistic decoding, incurring high latency. Past research (Liu et al. 2014) has shown that by caching the erroneous locations in flash memory blocks, the performance overhead of LDPC decoding can be significantly reduced since the error locations do not change significantly across reads until the block is erased. To this end, erroneous memory locations can be efficiently recorded in a base-offset form using ECS/XECS to reduce the LDPC decoding latency and parity overhead.

Finally, similar to PCM, NAND-flash-based SSDs are also vulnerable to permanent cell wearout failures, though the wearout rate of MLC/TLC NAND flash is $\approx 100\times$ higher than PCM (Cai et al.

2013; Meza et al. 2015a; Schroeder et al. 2016; Luo et al. 2016; Fukami et al. 2017; Cai et al. 2017). To reduce these wearout failures, modern NAND-flash-based SSD controllers employ data compression, data scrambling, and wear leveling to reduce cell writes (Schroeder et al. 2016; Cai et al. 2017). Further, these wearout failures grow with time, and the blocks with a large number of wearout failures (growth bad blocks, i.e., GBBs) are discarded by the SSD controller, reducing the SSD storage capacity. This presents an opportunity to integrate XECS with NAND-flash-based SSDs, since XECS can leverage the compression and wear-leveling architecture of SSDs to extend the usable lifetime of the GBBs. Whereas a detailed analysis of ECS-/XECS-based NAND flash architectures is beyond the scope of this work, it is an open area for future research.

8 RELATED WORK

Initial solutions to counter hard errors in NVMs included conventional ECCs (e.g., SEC-DED, BCH, etc.), followed by ECC-based data inversion (Maddah et al. 2013) and its extension as symbol shifting (Maddah et al. 2016). These techniques encode data to reduce the number of errors per write and consequently improve the strength of the underlying ECC. DRM (Ipek et al. 2010) was the first departure from ECC-based hard error recovery in NVMs. DRM recovers from errors by combining two faulty pages such that there are no common faulty positions in both the pages. DRM was followed by FREE-p (Yoon et al. 2011), which performs error recovery through fine-grained remapping of a faulty data block (64 bytes) to a new memory location and storing the remapping pointer in the failed block's position. By implementing the error correction codec inside the processor-side memory controller, FREE-p enables error correction/detection of errors in wires, packaging, and peripheral circuits in addition to errors in NVM cells.

Since DRM and FREE-p require special support from the OS to manage the pairing of the dead pages and remapping of the failed memory blocks, respectively, implementing these techniques increases the complexity and cost of the computing system. This led to the development of ECP (Schechter et al. 2010), which employs a pointer to point to a failed memory cell for error recovery, and incurs low performance/hardware overhead and does not require changes to or support from the OS. Due to this, ECP is considered a baseline reference for hard error correction in NVMs. State-of-the-art PAYG (Qureshi 2011) and Zombie memory (Azevedo et al. 2013) are ECP extensions that improve ECP utilization to further extend memory lifetime. PAYG follows on-demand ECP allocation policy to mitigate ECP wastage associated with uniform ECP allocation. In contrast, Zombie memory recycles the dead memory pages and uses them for storing additional ECPs.

Whereas ECC- and ECP-based techniques use redundancy and pointers, respectively, to recover from cell failures, techniques like SAFER (Seong et al. 2010b), RDIS (Melhem et al. 2012), and AEGIS (Fan et al. 2013) continue to use faulty (i.e., stuck-at) cells by leveraging the observation that a stuck-at cell can be used to store data if the stuck-at value of the faulty cell and the data bit to be written to that cell are equivalent. Although SAFER/RDIS/AEGIS are complementary to ECC- and ECP-based techniques, integrating SAFER/RDIS/AEGIS with ECC-based and/or ECP-based techniques increases the memory overhead of error correction. Furthermore, since these techniques rely on data inversion to recover from stuck-at faults, they render DCW ineffective, incurring high write energy and memory wear.

The main drawbacks of ECC- and ECP-based error correction techniques can be summarized as follows. First, techniques that rely on ECC (Lin and Costello 2004; Maddah et al. 2013, 2016) have to update the error-correcting bits whenever the data protected by those bits changes. This increases the wear of the cells in which error-correcting bits are stored. Also, since the number of hard errors increases with time, stronger codes are required for error recovery. Second, techniques like ECP and FREE-p (Yoon et al. 2011) that uniformly allocate error correction resources to all the rows in a page result in poor utilization of the error correction resources. Finally, techniques like DRM (Ipek

et al. 2010), FREE-p (Yoon et al. 2011), PAYG (Qureshi 2011), and Zombie memory (Azevedo et al. 2013) incur a high latency penalty (one to two additional reads for every read/write in the worst case) for error recovery.

Although XECS is similar to ECP since XECS also employs pointers to recover from cell failures, XECS significantly differs from ECP and ECP-based techniques as follows: (1) XECS follows a base-offset approach for recording the cell failures, which reduces the memory overhead of ECS storage; (2) XECS performs dynamic on-demand allocation of ECS pointers, improving the utilization of error correction resources; (3) XECS efficiently integrates data compression to accommodate more ECSs for extended error correction; and (4) XECS does not reduce performance below 2.5%, even in the presence of a high error rate.

In addition to hard error correction, several hard error prevention techniques (Yang et al. 2007; Cho and Lee 2009; Jacobvitz et al. 2013; Qureshi et al. 2009a; Seong et al. 2010a; Palangappa and Mohanram 2015; Palangappa et al. 2016) have been proposed in the literature. These techniques are complementary to hard error correction techniques and focus on reducing memory wear to delay the onset of hard errors. These techniques can be broadly classified into two categories: (1) write reduction techniques (e.g., DCW (Yang et al. 2007), Flip-N-Write (Cho and Lee 2009), Coset coding (Jacobvitz et al. 2013), Flip-Mirror-Rotate (Palangappa and Mohanram 2015), WOM-Code (Palangappa et al. 2016), etc.) and (2) wear-leveling techniques (e.g., Start-Gap (Qureshi et al. 2009a), Security-refresh (Seong et al. 2010a), application-specific wear leveling (Liu et al. 2013), warranty-aware wear leveling (Cheng et al. 2016), etc.). Additionally, techniques like Mellow writes (Zhang et al. 2016) and Region Retention Monitor (Zhang et al. 2017) perform slower writes to dissipate lower power for improved NVM lifetime. XECS integrates bit-level DCW to reduce memory wear and Start-Gap wear leveling to uniformly distribute writes across memory.

Furthermore, parts of XECS have similarities with (1) variable-strength ECC (VS-ECC) (Alameldeen et al. 2011) (dynamic error correction), (2) COP (Palframan et al. 2015) and Frugal ECC (Kim et al. 2015) (leveraging data compression for extended error correction), and (3) toggle-aware compression for GPU (Pekhimenko et al. 2016) (handling high entropy of the compressed data). VS-ECC (Alameldeen et al. 2011) was proposed as a dynamic on-demand error correction technique for caches. Since caches are volatile, VS-ECC invokes a special runtime cache characterization phase (which takes up to 50 seconds) to identify weak lines in a cache after every reboot. In contrast, XECS identifies the weak rows during a write, without requiring any additional characterization, and stores this information in ECS metadata on the NVM itself.

COP (Palframan et al. 2015) and Frugal ECC (Kim et al. 2015) improve soft error correction capabilities of DRAM-based main memories by leveraging data compression for storing ECC bits. However, these techniques cannot be employed for hard error correction in NVMs because these techniques are proposed for DRAM soft errors, which occur at a rate that is more than 2 orders of magnitude lower in comparison to the hard error rate in NVMs (Schroeder et al. 2009; Qureshi 2011). Due to this, the memory reserved for error correction by these techniques is insufficient for storing strong ECCs required for hard error correction in NVMs.

Finally, toggle-aware compression for GPUs (Pekhimenko et al. 2016) reduces energy while transmitting high entropy compressed data on GPU communication channels. A GPU communication channel transfers 128-byte data in four 32-byte flits. As a result, even if the width of the compressed data is less than 32 bytes, a 32-byte flit is transferred on the channel, increasing the BFP, that is, toggling of the channel. However, since XECS employs horizontal wear leveling, writing a compressed cache line to memory actually reduces the BFP (i.e., memory wear), because the wear is equally shared by all the cells storing the cache line. Furthermore, since Pekhimenko et al. (2016) is proposed for GPUs, Pekhimenko et al. (2016) ignores zero cache lines from its analysis because zero cache lines are already efficiently handled by modern GPUs (already clarified

in Pekhimenko et al. (2016)). However, since XECS focuses on NVM-based main memories, we do not exclude zero cache lines in our analysis. Since zero cache lines form a major part (25%) of the compressed data (with compression width of 3 bits, i.e., BFP of 0.0058), XECS significantly benefits from the high compressibility of zero cache lines.

For the latest trends and future research directions in NVM reliability, readers can also refer to other insightful surveys like Swami and Mohanram (2017), Mittal (2017), and Cai et al. (2017).

9 CONCLUSIONS

This article proposed error-correcting strings (ECSs), which adopt a base-offset approach to record the addresses of the failed memory cells for hard error correction in NVMs. ECS uses variable-length offsets to point to the failed cells, which accommodates more error-correcting pointers and increases the ability to tolerate more hard errors per memory block. To reduce the decoding overhead of variable-length offsets, we proposed row-level ECS that uses fixed-length offsets within a row, but allows different rows in page to have different offset lengths. This is a middle-ground solution to derive the maximum benefits of ECS while also simplifying the process of decoding the offsets for error correction.

This article also proposed eXtended-ECS (XECS), a page-level error correction architecture that employs dynamic on-demand ECS allocation and opportunistic pattern-based data compression to improve NVM lifetime by 2× over state-of-the-art ECP-6 for comparable overhead and negligible impact to system performance. Further, this article presented a comprehensive evaluation to demonstrate that XECS achieves higher NVM lifetime for lower memory overhead in comparison to state-of-the-art page-level error correction architectures. Finally, this article demonstrated that ECS is a drop-in replacement for ECP in state-of-the-art ECP-based techniques like PAYG and Zombie memory; ECS is also complementary to soft error correction techniques based on IDM and ECC for MLC/TLC NVMs.

REFERENCES

2011. *International Technology Roadmap for Semiconductors*.
- Alaa R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S. L. Lu. 2011. Energy-efficient cache design using variable-strength error-correcting codes. In *Proc. Intl. Symposium on Computer Architecture*.
- Alaa R. Alameldeen and David A. Wood. 2004. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. Department of Computer Science, University of Wisconsin-Madison, Technical Report.
- Manu Awasthi, Manjunath Shevgoor, Kshitij Sudan, Bipin Rajendran, Rajeev Balasubramonian, and Viji Srinivasan. 2012. Efficient scrub mechanisms for error-prone emerging memories. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.
- Rodolfo Azevedo, John D. Davis, Karin Strauss, Parikshit Gopalan, Mark Manasse, and Sergey Yekhanin. 2013. Zombie memory: Extending memory lifetime by reviving dead blocks. In *Proc. Intl. Symposium on Computer Architecture*.
- Geoffrey W. Burr, Matthew J. Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi, Chung Lam, Luis A. Lastras, Alvaro Padilla, Bipin Rajendran, Simone Raoux, and Rohit S. Shenoy. 2010. Phase change memory technology. *Journal of Vacuum Science and Technology B* 28 (2010), 223–262.
- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. 2017. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE* (2017).
- Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. 2013. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE'13)*.
- Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015a. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *Proc. Intl. Conference on Dependable Systems and Networks*.
- Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. 2015b. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *Proc. Intl. Symposium on High Performance Computer Architecture*.
- Sheng-Wei Cheng, Yuan-Hao Chang, Tseng-Yi Chen, Yu-Fen Chang, Hsin-Wen Wei, and Wei-Kuan Shih. 2016. Efficient warranty-aware wear leveling for embedded systems with PCM main memory. *IEEE Transactions on Very Large Scale Integration Systems* 24 (2016), 2535–2547.

- Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proc. Intl. Symposium on Microarchitecture*.
- Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y. J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y. T. Lee, J. Yoo, and G. Jeong. 2012. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'12)*.
- Jie Fan, Song Jiang, Jiwei Shu, Youhui Zhang, and Weimin Zhen. 2013. Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory. In *Proc. Intl. Symposium on Microarchitecture*.
- Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé. 2010. Increasing PCM main memory lifetime. In *Proc. Design, Automation & Test in Europe Conference & Exhibition*.
- Aya Fukami, Saugata Ghose, Yixin Luo, Yu Cai, and Onur Mutlu. 2017. Improving the reliability of chip-off forensic analysis of NAND flash memory devices. *Digital Investigation* (2017).
- John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. In *ACM SIGARCH Computer Architecture News*.
- Engin Ipek, Jeremy Condit, Edmund B. Nightingale, Doug Burger, and Thomas Moscibroda. 2010. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *Proc. Architectural Support for Programming Languages and Operating Systems*.
- A. N. Jacobvitz, R. Calderbank, and D. J. Sorin. 2013. Coset coding to extend the lifetime of memory. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.
- Lei Jiang, Youtao Zhang, and Jun Yang. 2012. ER: Elastic RESET for low power and long endurance MLC based phase change memory. In *Proc. Intl. Symposium on Low Power Electronics and Design*.
- Sachhith Kannan, Jeyavijayan Rajendran, Ramesh Karri, and Ozgur Sinanoglu. 2013. Sneak-path testing of crossbar-based non-volatile random access memories. *IEEE Transactions on Nanotechnology* 12 (2013), 413–426.
- Jungrae Kim, Michael Sullivan, Seong-Lyong Gong, and Mattan Erez. 2015. Frugal ECC: Efficient and versatile memory error protection through fine-grained compression. In *Proc. Intl. Conference for High Performance Computing, Networking, Storage and Analysis*.
- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable DRAM alternative. In *Proc. Intl. Symposium on Computer Architecture*.
- Shu Lin and Daniel J. Costello. 2004. *Error Control Coding*. Pearson Higher Education.
- Duo Liu, Tianzheng Wang, Yi Wang, Zili Shao, Qingfeng Zhuge, and Edwin Sha. 2013. Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems. In *Proc. Intl. Asia and South Pacific Design Automation Conference*.
- Ren-Shuo Liu, Meng-Yen Chuang, Chia-Lin Yang, Cheng-Hsuan Li, Kin-Chu Ho, and Hsiang-Pang Li. 2014. EC-Cache: Exploiting error locality to optimize LDPC in NAND flash-based SSDs. In *Proc. Design Automation Conference*.
- Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2016. Enabling accurate and practical online flash channel modeling for modern MLC NAND flash memory. *IEEE Journal on Selected Areas in Communications* 34 (2016), 2294–2311.
- Rakan Maddah, Sangyeun Cho, and Rami Melhem. 2013. Power of one bit: Increasing error correction capability with data inversion. In *Proc. Intl. Pacific Rim Symposium on Dependable Computing*.
- Rakan Maddah, Sangyeun Cho, and Rami Melhem. 2016. Symbol shifting: Tolerating more faults in PCM blocks. *IEEE Transactions on Computing* 65 (2016), 2270–2283.
- Rami Melhem, Rakan Maddah, and Sangyeun Cho. 2012. RDIS: A recursively defined invertible set scheme to tolerate multiple stuck-at faults in resistive memory. In *Proc. Intl. Conference on Dependable Systems and Networks*.
- Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. 2015a. A large-scale study of flash memory failures in the field. In *ACM SIGMETRICS Performance Evaluation Review*.
- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. 2015b. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *Proc. Intl. Conference on Dependable Systems and Networks*.
- Sparsh Mittal. 2017. A survey of soft-error mitigation techniques for non-volatile memories. *Computers* (2017).
- Dimin Niu, Qiaosha Zou, Cong Xu, and Yuan Xie. 2013. Low power multi-level cell resistive memory design with incomplete data mapping. In *Proc. Intl. Conference on Computer Design*.
- Poovaiah M. Palangappa, Jiayin Li, and Kartik Mohanram. 2016. WOM-Code solutions for low latency and high endurance in phase change memory. *IEEE Transactions on Computing* 4 (2016), 1025–1040.
- Poovaiah M. Palangappa and Kartik Mohanram. 2015. Flip-Mirror-Rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories. In *Proc. Great Lakes Symposium on VLSI*.
- Poovaiah M. Palangappa and Kartik Mohanram. 2016. CompEx: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVM. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.

- David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. 2015. COP: To compress and protect main memory. In *Proc. Intl. Symposium on Computer Architecture*.
- Avadh Patel, Furat Afram, and Kanad Ghose. 2011. Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors. In *Proc. Design Automation Conference*.
- G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler. 2016. A case for toggle-aware compression for GPU systems. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.
- Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2012. Base-delta-immediate compression: Practical data compression for on-chip caches. In *Proc. Intl. Conference on Parallel Architectures and Compilation Techniques*.
- Moinuddin Qureshi. 2011. Pay-As-You-Go: Low-overhead hard error correction for phase change memories. In *Proc. Intl. Symposium Microarchitecture*.
- Moinuddin Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009a. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proc. Intl. Symposium on Microarchitecture*.
- Moinuddin Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009b. Scalable high performance main memory system using phase change memory technology. In *Proc. Intl. Symposium on Computer Architecture*.
- Stuart Schechter, Gabriel H. Loh, Karin Straus, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *Proc. Intl. Symposium Computer Architecture*.
- Bianca Schroeder and Garth Gibson. 2010. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* 7 (2010), 337–350.
- Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. 2016. Flash reliability in production: The expected and the unexpected. In *USENIX Conference on File and Storage Technologies*.
- Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM errors in the wild: A large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*.
- Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. 2010a. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. *Proc. Intl. Symposium on Computer Architecture* (2010).
- Nak Hee Seong, Dong Hyuk Woo, Vijayalakshmi Srinivasan, Jude A. Rivers, and Hsien-Hsin S. Lee. 2010b. SAFER: Stuck-at-fault error recovery for memories. In *Proc. Intl. Symposium on Microarchitecture*.
- Nak Hee Seong, Sungkap Yeo, and Hsien-Hsin S. Lee. 2013. Tri-level-cell phase change memory: Toward an efficient and reliable memory system. In *Proc. Intl. Symposium on Computer Architecture*.
- Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory errors in modern systems: The good, the bad, and the ugly. In *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*.
- Shivam Swami and Kartik Mohanram. 2016. E³R: Energy efficient error recovery for multi/triple-Level cell non-volatile memories. In *Proc. Intl. Conference on VLSI Design*.
- Shivam Swami and Kartik Mohanram. 2017. Reliable non-volatile memories: Techniques and measures. *IEEE Design & Test* (2017).
- Cong Xu, Dimin Niu, Naveen Muralimanohar, Norman P. Jouppi, and Yuan Xie. 2013. Understanding the trade-offs in multi-level cell ReRAM memory design. In *Proc. Design Automation Conference*.
- Chun Jason Xue, Youtao Zhang, Yiran Chen, Guangyu Sun, J. Jianhua Yang, and Hai Li. 2011. Emerging non-volatile memories: Opportunities and challenges. In *Proc. Intl. Conference on Hardware/Software Codesign and System Synthesis*.
- Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. 2007. A low power phase change random access memory using a data-comparison write scheme. In *Proc. Intl. Symposium on Circuits and Systems*.
- Doe Hyun Yoon, Jichuan Chang, Robert S. Schreiber, and Norman P. Jouppi. 2013. Practical non-volatile multi-level cell phase change memory. In *Proc. Intl. Conference on High Performance Computing*.
- Doe Hyun Yoon, Naveen Muralimanohar, Jichuan Chang, Parthasarathy Ranganathan, Norman P. Jouppi, and Mattan Erez. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.
- Vinson Young, Prashant Nair, and Moinuddin Qureshi. 2015. DEUCE: Write-efficient encryption for non-volatile memories. In *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*.
- Jianhui Yue and Yifeng Zhu. 2013. Accelerating write by exploiting PCM asymmetries. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.
- Lunkai Zhang, Brian Neely, Diana Franklin, Dmitri Strukov, Yuan Xie, and Frederic T. Chong. 2016. Mellow writes: Extending lifetime in resistive memories through selective slow write backs. In *Proc. Intl. Symposium on Computer Architecture*.
- Mingzhe Zhang, Lunkai Zhang, Lei Jiang, Zhiyong Liu, and Frederic T. Chong. 2017. Balancing performance and lifetime of MLC PCM by using a region retention monitor. In *Proc. Intl. Symposium on High-Performance Computer Architecture*.

- Wangyuan Zhang and Tao Li. 2009a. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proc. Intl. Symposium on Microarchitecture*.
- Wangyuan Zhang and Tao Li. 2009b. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proc. Intl. Conference on Parallel Architectures and Compilation Techniques*.
- Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *Proc. Intl. Symposium on Computer Architecture*.

Received March 2017; revised September 2017; accepted September 2017