Block Cooperation: Advancing Lifetime of Resistive Memories by Increasing Utilization of Error Correcting Codes

MOHAMMAD KHAVARI TAVANA, AMIR KAVYAN ZIABARI, and DAVID KAEI,
Northeastern University, USA

Block-level cooperation is an endurance management technique that operates on top of error correction mechanisms to extend memory lifetimes. Once an error recovery scheme fails to recover from faults in a data block, the entire physical page associated with that block is disabled and becomes unavailable to the physical address space. To reduce the page waste caused by early block failures, other blocks can be used to support the failed block, working cooperatively to keep it alive and extend the faulty page's lifetime.

We combine the proposed technique with existing error recovery schemes, such as Error Correction Pointers (ECP) and Aegis, to increase memory lifetimes. Block cooperation is realized through metadata sharing in ECP, where one data block shares its unused metadata with another data block. When combined with Aegis, block cooperation is realized through reorganizing data layout, where blocks possessing few faults come to the aid of failed blocks, bringing them back from the dead.

Our evaluation using Monte Carlo simulation shows that block cooperation at a single level (or multiple levels) on top of ECP and Aegis, boosts memory lifetimes by 28% (37%) and 8% (14%) on average, respectively. Furthermore, using trace-driven benchmark evaluation shows that lifetime boost can reach to 68% (30%) exploiting metadata sharing (or data layout reorganization).

CCS Concepts: • **Computer systems organization** → **Processors and memory architectures**; *Reliability*; • **Hardware** → **Non-volatile memory**; **Memory and dense storage**;

Additional Key Words and Phrases: Resistive memory, phase change memory, memory reliability, error correction, error recovery, workload characterization, endurance

ACM Reference format:

Mohammad Khavari Tavana, Amir Kavyan Ziabari, and David Kaeli. 2018. Block Cooperation: Advancing Lifetime of Resistive Memories by Increasing Utilization of Error Correcting Codes. *ACM Trans. Archit. Code Optim.* 15, 3, Article 36 (August 2018), 26 pages.

<https://doi.org/10.1145/3243906>

Extension of Conference Paper; Mohammad Khavari Tavana, Amir Kavyan Ziabari, and David Kaeli. 2017. Live together or die alone: Block cooperation to extend lifetime of resistive memories. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE'17)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 1098-1103.

Authors' addresses: M. K. Tavana, A. K. Ziabari, and D. Kaeli, Northeastern University, 360 Huntington Ave, Boston, MA, 02115; emails: {mkhavaritavana, aziabari, kaeli}@ece.neu.edu.

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.

© 2018 Association for Computing Machinery.

1544-3566/2018/08-ART36 \$15.00

<https://doi.org/10.1145/3243906>

1 INTRODUCTION

Due to shrinking cell sizes in successive technology nodes, DRAM scaling has become a challenging problem [20]. Limited electrical charge that can be trapped in the DRAM capacitors, reduces retention time and dramatically increases the chance of unreliable sensing [20, 26]. Moreover, design complexity and manufacturing barriers become more significant when working below a 20nm technology node [20].

Given the proliferation of cloud computing and big data analytics, next-generation workloads will require higher memory capacity to be able to store significant amounts of data closer to the processor. DRAM technology is struggling to meet these requirements. Instead, resistive memories have been proposed as a viable alternative or complement [28].

Instead of representing the bit state as an indicator of the presence or absence of electrical charge, resistive memories work by measuring the resistivity of cells. The cell resistivity is determined by the atomic arrangement of the memory material. Some examples of resistive memory technologies are Phase-Change Memory (PCM), Ferroelectric RAM (FRAM), Spin Transfer Torque Memory (STT-MRAM), and memristors. PCM is the most promising of these choices for main memory due to its maturity and the advances made in technology over the past decade. PCM has a number of desirable characteristics, including zero leakage power, comparable performance to DRAM, and the ability to retain cell state for more than 10 years [20]. These features pave the way for PCM to be mass produced and commercialized [2, 11]. Furthermore, given that recent devices support multiple bits per cell in PCM [2], this can double or triple the memory density.

Despite these desirable features, write endurance still looms as the major drawback of using PCM [20]. PCM cells lose their ability to switch between states once they pass their endurance limits. This results in either a stuck-at “0” or “1” fault. Such errors are permanent and cumulative, so will result in significant degradation in system reliability. Each type of PCM cell and technology supports a specific write endurance limit. Single-Level Cell (SLC) PCM can tolerate around 10^7 – 10^9 writes, while Multi-Level Cell (MLC) and Triple-Level Cell (TLC) PCM can tolerate 10^6 write cycles [2, 41, 51].

To mitigate the write endurance problem, proactive approaches can be employed to reduce the number of writes or bit flips in the PCM. Data Comparison Write (DWC) is used in the PCM chip to limit the cell writes to only modified bits [52]. The Flip-N-Write technique [9] is an improvement over DWC to write memory cells in their inverted form if it causes fewer bit flip. Data encoding is another approach that can reduce the number of bit flips significantly [31, 43]. At memory block level, shifting and rotating are employed to uniformly distribute bit flips within a block [35, 52]. Other mechanisms, such as start-gap [39], has been proposed to distribute writes uniformly among physical pages. Despite employing proactive mechanisms that attempt to prolong PCM lifetime, Error-Correcting Codes (ECC) are needed in order to tolerate errors in the case of cell failure. Extra metadata, stored in memory along with data (in the ECC bits) can assist in error recovery, increasing memory lifetimes.

A number of error correction schemes have been proposed that can recover from stuck-at faults in resistive memories and extend memory lifetimes [4, 16, 32, 38, 41, 42, 49]. However, more reliable and robust memory systems are always needed by enterprise grade hardware/software systems. Additionally, in secure systems, where data are stored encrypted in main memory, resistive memory lifetimes can be reduced by up to an order of magnitude [44, 50] due to high entropy of the encrypted data bits. To increase the chances of future adoptions of resistive memories by industry, they will need to be able to tolerate more cell failures.

Although there are several techniques to evenly distribute the number of writes across all the memory cells, given reliability issues with shrinking technology nodes and the associated manufacturing challenges [51], as well non-uniform write patterns in many applications [38–40, 47],

Fig. 1. Memory lifetime and the average error per page for different scenarios. Each scenario mimics different behavior of workloads or memory system. Non-uniformity in write accesses reduces PCM memory lifetime significantly and results in under-utilization of error correcting chip for error recovery.

error correction schemes are less effective. To show the impact of non-uniformity on memory lifetimes, we model more than two thousand 4K pages, each containing 64 memory blocks.¹ We define five scenarios with generating different synthetic workloads and PCM parameters. Each scenario is described as follows:

- (1) We assume each block experiences 50% bit flips in each write, and the flips are uniformly distributed across cells. We also assume perfect wear leveling and uniform writes across all memory blocks. The mean value of cell lifetimes is assumed 10^8 , which follows a normal distribution, with a dispersion of 20% around the mean value (i.e., Coefficient of Variance or CoV = 0.2).
- (2) This scenario represents the impact of process variation on the lifetime of cells. This scenario is similar to scenario 1, except we increase the Coefficient of Variance (CoV) to 0.3.
- (3) Workloads exhibit a different number of bit flips during each memory writeback. To synthetically reproduce this behavior, we randomly flips 10–50% of cells at each write. The other parameters are similar to scenario 1.
- (4) Workloads usually do not write to all memory blocks of all the pages. To reproduce this level of non-uniformity in workloads, we select between 1 to 32 blocks to write in each page versus writing all blocks. The other parameters are similar to scenario 1.
- (5) In this scenario, we combine all the previous non-uniformity in workloads and process technology. The CoV is assumed to be 0.3, bit flips per writes are 10–50%, and we only select 1 to 32 blocks of 64 blocks in each page to be written.

We model Error Correction Pointer (ECP) [41] and Aegis [16], two state-of-the-art error correction schemes proposed for resistive memories. Once an error correction scheme can no longer tolerate faults in a data block, the entire physical page associated with that data block must be disabled and becomes unavailable to the physical address space. We continuously write to the memory and record the number of errors in each data block until all the pages are taken offline. In Figure 1, we plot the memory lifetime in terms of billion writes per page (Y-axis) versus the average number of errors per page (X-axis) for both ECP and Aegis. Each point represents a specific scenario for synthetic workloads.

¹We describe our simulation methodology in detail in Section 5.1.

By stressing the memory with different scenarios, we make the following observations:

- (1) Non-uniformity in write accesses, as well as the manufacturing process, can dramatically reduce the memory lifetime. Note that the non-uniformity increases gradually from scenario 1 to scenario 5. Using the number of writes per page (Y-axis) as a proxy for memory lifetime, we can impact wearout by a factor of 8X and 6X for ECP and Aegis, respectively, in the worst case.
- (2) Even though ECP can tolerate up to six faults per data block, almost 77% of data blocks experience three or fewer faults in scenario 1. However, Aegis² can tolerate 10 faults (deterministically) and is able to tolerate more faults (probabilistically). However, almost 50% of all of the data blocks tolerate five or fewer faults. Increasing non-uniformity severely reduces the effectiveness of using error correcting codes. We see a shift to the left when compare the average number of errors per page occurring, as we compare scenarios 1 through 5. The average number of corrected errors per page is reduced by a factor of 4X for both error correcting schemes when experiencing increased non-uniformity.

The observations for these two different error recovery schemes confirms that non-uniformity hampers the effectiveness of error correction and reduces the utility of the ECC bits, significantly. Informed by these observations, we propose using *block cooperation*, a simple but effective technique that can be incorporated with different error correction schemes to boost PCM lifetimes, while imposing little extra cost.

To evaluate the effectiveness of our proposed approach, we exploit block cooperation on top of ECP [41] and Aegis [16] error correction schemes. Block cooperation allows memory blocks that experience a small number of errors to come to the rescue of blocks with a larger number of errors, avoiding an early page death due to non-uniform writes. Block cooperation is realized through metadata sharing in ECP. Metadata sharing can be performed at a single level, where one data block shares its unused metadata with another data block, or can be implemented to be multi level, where multiple data blocks can share their metadata together. In Aegis, block cooperation is realized through data layout reorganization, where the blocks with fewer faults can help failed blocks to bring a page back to life. Using single-level (or multi-level) block cooperation, we can increase memory lifetimes by 28% (37%) and 8% (14%), on average, for ECP and Aegis, respectively.

This article substantially extends our previous work [47], where we proposed the basic idea of block cooperation, as follows:

- (1) We provide a comprehensive workload characterization by analyzing write access patterns that impact memory lifetimes. In this phase, we describe and determine fundamental workload execution metrics that most directly impact memory lifetimes. We identify key benchmark features as part of our workload characterization and use these quantitative metrics to make choices that impact memory lifetimes. Our characterization uses trace-driven simulation that models the data values transferred between on-chip and main memories.
- (2) In previous work [47], only statistical simulation is performed to show the benefits of block cooperation. This approach has been followed in previous lifetime studies [16, 30, 41]. In this article, we extend our lifetime analysis framework to simulate write accesses at a bit-level granularity. We model not only the writes to memory blocks but also data values to capture the number of bit flips during each writeback for more accurate lifetime analysis. Moreover, start-gap [39] is integrated into our framework as a low-cost and state-of-the-art wear-leveling mechanism to efficiently distribute writes at a block level.

²The configuration for Aegis is considered 17×31 [16].

Fig. 2. Standard PCM-based dual in-line memory module (DIMM) memory interface with ECC chip.

The rest of this article is organized as follows: In Section 2, we briefly present related work, reviewing error detection and failure mechanisms in PCM. We present our block cooperation scheme in Section 3. Metadata sharing for ECP, and data layout reorganization for Aegis, are covered in Section 4. Section 5 describes our experimental framework for both Monte Carlo and trace-driven simulation, workload characterization, and simulation results. Finally, in Section 6, we summarize the lessons learned in this article.

2 BACKGROUND AND RELATED WORK

To mitigate the write endurance problem in PCMs, prior work proposed techniques to proactively alleviate the write impact on cell endurance. Fine-Grained Current Regulation (FGCR) [23], which is possible by modifying the PCM's peripheral circuit, reduces the RESET programming power and increases memory lifetime. Employing a DRAM buffer [28], data coding/encoding [4, 33, 45], wear-leveling algorithms [1, 39], and data compression [5, 22, 36] are among the several competing mechanisms that have been explored to improve PCM lifetime. Although these techniques are effective in prolonging memory lifetimes, error correction mechanisms still need to be included to handle faults due to wear-out.

In this section, first we describe write mechanisms and fault detection mechanisms used in PCM Dual In-line Memory Modules (DIMM). Then, we discuss the different fault models in PCM. Finally, we review prior work targeting hard error correction in PCM.

2.1 Write Mechanisms in PCM DIMMs

Since write operations are costly in PCM and result in cell wear-out, and only a limited portion of the memory block is modified on each writeback to main memory [44, 50], *Data Comparison Writes (DCWs)* [52] are used to limit bit writes to only the differences in a memory block. Figure 2 shows a PCM DIMM. More specifically, a read-modify-write circuit [52] is provided on PCM chips to first read the old data, find the differences with the new data, and only write to the modified bits in the memory block. A PCM rank comprises nine chips, where the ninth chip stores an Error Correcting Code (ECC) for the other eight data chips. The data is interleaved across all of the chips in the DIMM. In the case of reading/writing from/to erroneous blocks, metadata in the ECC chip is exploited for error recovery. Therefore, for every 64-byte data block, 8 bytes of storage are used to maintain the metadata. The standard DIMM has a 72-bit data path (along with 8 bits for ECC chip), therefore eight burst accesses are required to read or write a 64-byte block of data.

In cases of stuck-at faults, the cell cannot be reprogrammed but can be read using normal read circuitry in PCMs. Therefore, faults can be detected on write accesses using read verification. Read verification circuitry [2, 10, 25] is a standard part of the PCM technology, embedded into the logic of PCM chips. The extra read ensures that the resistance of each cell is within a safe region and

is used for error detection as well. Read verification mechanism is used for error detection in our work (as considered in prior studies [4, 16, 30, 32, 38, 41, 42, 45, 46, 49]).

2.2 PCM Fault Model

2.2.1 Hard Faults. PCM stores data in either a low-resistance crystalline state (SET or “1”) or a high-resistance amorphous state (RESET or “0”). As more and more state transitions occur in a PCM cell, at some point the cell loses its ability to switch and the cell will be stuck at either a SET or a RESET state. The former is referred to as a SET-stuck failure (SSF), and the latter is referred to RESET-stuck failure (RSF), respectively.

SSF and RSF occur for different reasons. An SSF occurs due to germanium (Ge) depletion inside the programmable area, which affects the level of cell resistance [27]. However, an RSF happens because of the detachment of the heating electrode used for programming [27]. SSF can be recovered to some extent by utilizing a reverse electric field [29], while RSF is unrecoverable. We refer to both as hard faults or stuck-at faults in this article. These errors are permanent and only occur after write operations.

2.2.2 Soft Faults. Soft faults are not the focus of this article. Nonetheless, they should be mitigated to design a robust main memory system. The sources of these faults in PCM are different as compared to DRAM and can be classified as follows:

(i) *Resistance drift faults.* Resistance drift is not a problem for Single-Level-Cell (SLC) PCM. However, in Multiple-Level-Cell (MLC) PCM, cell resistance increases over time and there is a chance that intermediate states crossing the resistance boundary, consequently resulting in unreliable read-out. Different techniques have been proposed to tolerate resistance drift. We can exploit reference cells and data encoding [2, 37], as well as scrubbing mechanisms [3], to address drift faults in MLC.

(ii) *Write disturbance faults.* When writing to a cell, adjacent cells may be thermally disturbed, and their resistance may change. This inadvertent resistance change when writing to neighboring cells in a PCM is referred to as a *write disturbance*. This problem is more pronounced in small device geometries and dense PCM, where the inter-cell distance can be very small [15, 24, 45, 48]. Trading off capacity for reliability by introducing thermal bands along cell lines can effectively reduce write disturbance. Data encoding techniques [15, 24, 45] are also an effective approach to reduce the frequency of write disturbance.

2.3 Related Work: Hard-Fault Tolerance in PCM

Hard errors are permanent and increase over time. So the mechanisms found in current DRAMs (such as Single Error Correction, Double Error Detection–SECDED) are insufficient for PCMs due to their very different error profiles. Prior work proposed error correction schemes specifically designed for PCM to improve memory lifetime. We classify these schemes into the following three categories [46].

Category 1 – Replacing Faulty Bits with Healthy Bits. The basic idea is to find the location of bit failures and to replace them with healthy bits. The Error Correction Pointer (ECP) scheme [41] is the pioneering work in this category, where for every faulty bit, the scheme keeps one pointer and one replacement bit (forming one ECP entry). Error correction is performed by restoring the correct value of the faulty bit (the location where the ECP points) with the replacement bit. Assuming a standard DIMM with one ECC chip for every eight data chips [21], ECP can correct six faulty bits, irrespective of the location of the faulty memory cells in a 64-byte data block. This is known as ECP6, and we will refer to it as such throughout this article.

Qureshi [38] proposes the Pay-As-You-Go (PAYG) error recovery scheme to significantly reduce the storage overhead of ECP. Leveraging PAYG, for each data block, one dedicated ECP entry is allocated. To correct more bit failures, PAYG maintains additional ECP entries in a global structure, enabling dynamic management and reducing the storage overhead. PAYG enjoys a 3× reduction in ECP storage overhead, with only minimal performance degradation.

REMAP [46] uses all the metadata space for replacing faulty bits. The locations of failed memory cells are identified by extra write operations, instead of storing a pointer to each failed cell. Therefore, the number of correctable errors increases. To alleviate the overhead of extra writes on memory lifetime, REMAP uses static and dynamic partitioning.

Category 2 – Partitioning Data Blocks and Bit-wise Inversion. The schemes in this category mask errors by storing data in their inverted forms (e.g., storing a “0” instead of a “1” into a stuck-at RESET cell). A necessary condition to be satisfied to use inversion for masking errors is that each faulty cell needs to be in a different partition. SAFER [42] partitions the data block dynamically to ensure that each partition has at most one faulty bit and decides whether to store data in its original or complement form. Aegis [16] exploits the same approach but uses a more efficient data block partitioning scheme—Aegis corrects more faulty bits with fewer partitions, as compared to SAFER. RDIS [30, 32] works in the same way but is more efficient than SAFER when the number of faulty bits grows beyond six. However, there are some error patterns that RDIS may fail to recover from when the number of errors is even greater than 3.

Compared to the schemes in the first category, all three implementations in the second category can tolerate more errors by removing the need for pointers in metadata. However, these schemes usually suffer from two issues. First, due to the lack of tracking the location, the solutions in the second category usually impose extra write overhead, which negatively impacts memory performance and endurance. The second issue relates to increased design complexity when implementing complicated partitioning algorithms.

Category 3 – Pairing the Faulty Blocks. When a memory block (either a data block or page) becomes faulty, instead of discarding it, we can pair it with another failed block to restore data correctness using redundancy. Dynamically Replicated Memory (DRM) [19] implements this concept by pairing two failed memory pages whose faulty bits are not at the same offset. Using this scheme, DRM delivers better memory system lifetime by gradually degrading the available storage. Free-p [49] is another scheme where each memory block that has many faulty bits stores a pointer to a replacement block, reducing the need for a pointer field as required in the first category. The pointer is replicated multiple times to tolerate errors. Zombie [4] extends the memory lifetime by reusing the healthy blocks of a discarded page that is visible to software. Zombie significantly improves endurance but requires complicated bookkeeping at a hardware level.

Chen et al. [8] proposed dynamic redundancy using mirroring and parity pages. In the mirroring scheme one redundant (mirror) bit is selected for every data bit by grouping two pages. The mirror page is selected in a way that there is no fault in the same bit location of the corresponding faulty page. In the parity page scheme, N faulty PCM pages form a group together to tolerate errors. This scheme imposes off-chip DRAM buffer and on-chip cache structure overheads for page mapping.

The schemes that are based on pairing exploit dead pages to tolerate errors. As a result, the visible memory space is reduced over time as more faults are encountered.

The main difference of the Block cooperation scheme with pairing schemes (category 3) is that it aims to provide improved fault-tolerance at a block level, without using spare memory blocks. If block cooperation is not able to correct errors, then pairing schemes can be applied as another layer of defense to resuscitate the page with help from another spare page.

Collaborative sharing of metadata increases metadata utilization in block cooperation and can be built on top of the schemes described in categories 1 and 2 to extend memory lifetime. We

3 BLOCK COOPERATION

In general, block cooperation can be performed on a single level, where only a single live block cooperates with another dying block, or can be performed across multiple levels, where multiple live blocks are able to cooperate with a single dying block. A control field is required in each block's metadata to maintain the state of the data blocks and any information that supports cooperation. An indirection pointer is used to help the memory controller quickly find a cooperating block or blocks.

At the highest level of the statechart hierarchy, the *data block status* contains two superstates, *faulty* and *non-faulty*. Once a cell becomes faulty due to wear-out, the state for the data block transitions from non-faulty to faulty. Within these two superstates, the following sub-states are defined:

- *Private*. The block does not cooperate with other blocks.
- *Shared*. The block cooperates and shares its data or metadata with another block.
- *Shared⁺*. The block shares its data or metadata with another block, and uses indirection pointer to refer to another shared or shared⁺ block.
- *Co-op*. The block is dead and has requested cooperation to become live again.

The following actions/operations are also defined to facilitate our cooperative scheme:

- *Join*. When a block needs to cooperate, a candidate block is selected, and then a join operation is initiated. The candidate block needs to be in the *private* state, and after joining, the state of the block changes to *shared*.
- *Disjoin*. A block in the *shared* or *shared⁺* state may not share its data or metadata for its own benefit. In this case, a disjoin operation is performed, and a transition to the *private* state is made. Since the corresponding block in the *Co-op* state has lost its block cooperator, a different data block(s) needs to be selected to join and save the entire page from failure.
- *Indirection*. When a block needs to cooperate with another block to survive, it transitions to the *Co-op* state, and the cooperator block transitions to the *shared* state. In case of multi-level cooperation, a data block may request cooperation with more blocks. Hence, an indirection operation is performed to have another block join the shared pool. This operation also sets an indirection pointer for the cooperating block. By adding indirection pointers to the metadata, the block can transition to the *shared⁺*.
- *Cooperation request*. Requests for cooperation are initiated when (i) a data block is in the *private* state and is about to die or (ii) during a disjoin or indirection operation, where a data block requests cooperation with more blocks. If the block selection policy finds a block that can resuscitate the failed block, then the data block transitions to the *Co-op* state (if it is not already in that state). Otherwise, it is no longer possible to save the block and the entire page must be disabled.

For block cooperation, the statechart can be simply implemented using a finite-state machine and reserving three bits of metadata in the ECC chip.

4 BLOCK COOPERATION ADOPTION

To show the effectiveness of our approach, we select two error correction schemes designed specifically for resistive memories: (i) ECP [41] and (ii) Aegis[16]. We integrate block cooperation with both of them. Next, we outline the range of design parameters considered in this work.

Block cooperation is integrated with ECP and Aegis using *metadata sharing* and *data layout re-organization* techniques, respectively. In the former, data blocks cooperate by sharing their metadata, while in the latter, blocks cooperate through data sharing and shuffling. In the following, we describe the details of block cooperation adoption.

4.1 Block Cooperation in ECP

The Error Correction Pointer (ECP) [41] is the pioneering work for using ECC in resistive memories. The faulty bits are located by a read verification after the write. Then for every faulty bit, one pointer and one replacement bit are kept in the metadata (forming one ECP entry). The metadata storage requirement increases by introducing more faults in a data block. Error correction is performed by replacing the correct bit with the bit pointed to by the pointer. In a standard DIMM with one ECC chip per eight data chip, 8-byte of metadata are available for a 64-byte data block, ECP provides the ability to correct up to six faults, which we will refer to as ECP6 throughout the rest of this article.

Fig. 4. ECP metadata format when using (a) single-level and (b) multiple-level metadata sharing.

In the ECP6 scheme, one bit is used to indicate whether a block is faulty or not, and 60 bits are used to keep track of the six ECP entries. As discussed in Section 3, three bits are required to maintain a block's status and enable metadata sharing for ECP. When a block is in the *private* state, the rest of the metadata (61 bits) provides six ECP entries. Therefore, the error correction capability (in the worst case) is not less than the standard ECP6.

Figure 4 shows the metadata format when single or multi-level sharing is used. When a block is in the *Co-op* state, six more bits are required for the indirection pointer. The indirection pointer field shows the location of helper block in page. Five bits indicate the number of extra ECP entries required by the Co-op block, thereby providing the capability of tolerating 36 faults for a block (in the best case). Providing block cooperation requires additional fields in the metadata, which results in reducing ECP entries to 5. In other words, we can only have six ECP entries if a block is in the *private* state, otherwise the number is reduced to 5.

When a block is in the *shared* or *shared⁺* state, three bits are required to track the number of ECP entries used. This field is essential for distinguishing between the ECPs that are used privately for a block, and the ECPs that are shared with other blocks. In the disjoin operation, the state of the current block, and the indirection pointer of a block that points to the current block need to be updated. In the *shared* state, six bits are allocated to indicate the borrower block. This field is not essential for the functionality of metadata sharing. However, the field provides a structure, similar to a circular list, that is used during the disjoin operation. In this way, the helped block can easily be reach to join it with another helper block.

The policy for selecting a block for join operations can be simply a random policy. However, the better selection policy is to find a block with no or the lowest number of faulty cells. To this end, all of the metadata in the block should be read, which incurs one (best-case) to 63 (worst-case) read(s) in the corresponding page. Note that this operation is not frequent, and aggregation of extra read latencies in the case of a row buffer hit in the page is small [21].

4.2 Block Cooperation in Aegis

Aegis [16], which similar to SAFER [42] and RDIS [30], uses partitioning and bit inversion to mask stuck-at errors by storing faulty values in their inverted form. To exploit fault masking in these

(a) Aegis with no block cooperation. No valid slope is found for block α , which results in disabling an entire page. With a slope equal to 2, four faulty cells can reside in four different partitions in the block β . For block γ , a slope of 0 is enough to tolerate one fault.

(b) Aegis with multiple block cooperation. Blocks β and γ are joined with the dead block α to save the entire page. Data layout reorganization is able to find a valid slope, in order to revive the dead block. Each portion of the plane (i.e., P_1 , P_2 and P_3) is filled by one of the three blocks.

Fig. 5. Integrating block cooperation in Aegis.

error correction schemes, faults should be isolated logically using partitioning. The goal of the partitioning scheme is to guarantee the existence of no two faults in the same partition.

Aegis partitioning distributes faults into different groups. When a new error is introduced in a group with an error, re-partitioning is performed to resolve error collisions. Aegis performs this partitioning by mapping data blocks to a two-dimensional (2D) Cartesian plane with $A \times B$ elements. On a Cartesian plane, any two different points on a line determines the slope of the line. Changing the slopes of the line preserves, at most, one cell of the original line to stay on a new line. All the cells that are covered by a line, belong to the same group. In the Aegis error correction scheme, there are B lines that share a common slope, so each cell can be covered by only one line or group. The re-partitioning is performed simply by changing the slope associated with a block. The slope can be stored with only $\lceil \log_2 B \rceil$ bits.

The plane for a simplified 32-bit data block, in a 5×7 organization, is shown in Figure 5(a). In this format, each error is shown as a black square on the plane, while partitions are the B parallel

lines (in this example B is equal to 7). The slope of the parallel lines should be chosen such that no two faults are placed on the same line or partition. If a valid slope (i.e., $0 \leq K < B$) is not found that can satisfy this property in a block, then error recovery is not possible.

To describe how block cooperation is adopted in Aegis with our simplified Cartesian plane, we show three different blocks in Figure 5(a). Blocks α , β , and γ contain six and four faults and one fault, respectively. No valid slope is found for block α , whereas blocks β and γ can tolerate the fault pattern with slopes of one and zero, respectively.

Similarly to ECP, we can exploit block cooperation, allowing multiple blocks to work cooperatively to keep their associated page alive. In contrast to ECP, Aegis metadata is fixed and independent of the number of faults in a data block. Therefore, it is not possible to share metadata to exploit underutilized metadata, so a different approach is needed for block cooperation.

To extend the memory lifetimes in Aegis, we use *data layout reorganization*, where blocks with fewer faults help dying blocks (i.e., blocks with no valid slope) to keep the page alive. For instance, in Figure 5(b), three blocks are joined together and each portion of the plane (i.e., P_1 , P_2 , and P_3 in Figure 5(b)) is filled by one of the three blocks. The new slopes for new data layout are calculated to check whether all the faults in the data blocks are recoverable. Note that the data layout reorganization is performed in the Aegis buffer in the memory controller. A block that requests cooperation transitions to the *Co-op* state, and the block that is selected to join transitions to the *shared* state. If we fail to find a valid slope for the new data layout, then more blocks are gradually invited to join the *Co-op* block to help. Block cooperation in Aegis helps to spread faults across different blocks more evenly, resulting in limited page waste due to early block failures. For example, in Figure 5(a), the number of faults are six, four, and one in the data blocks that change to four, four, and three in Figure 5(b), after applying data layout reorganization.

Assuming two blocks cooperate to save a faulty block, but no valid slope K is found for at least one of them, then the controller needs to select another private data block to perform the join operation. The number of retries that can be performed is equal to the number of private blocks within a page. We can also select a predefined *maximum try threshold*. For instance, the block α in Figure 5(a) cannot be saved, even if cooperates with the block β . However, for the next retry, if the block γ is selected, the block α can be revived.

In multi-block cooperation, new blocks can be joined with a *Co-op* block through indirection, until there are no more private blocks. Alternatively, a predefined limit can be set for the number of shared blocks to manage the complexity of error correction. We define this variable as the *sharing level* in the data layout reorganization. Note that in metadata sharing in ECP, accesses to *Co-op* blocks requires accesses to helper blocks for recovery, but helper blocks are self-contained and accessing them does not incur extra accesses to other blocks. In data layout reorganization, this is not the case, because for data recovery for both helper and helped blocks, issuing accesses to other blocks is necessary.

Figure 6 breaks down the required fields for the Aegis metadata when block cooperation is enabled. Just as with ECP, Aegis requires both a 3-bit Block status field and an indirection pointer field. Additionally, a $\lceil \log_2 B \rceil$ -bit field is required to keep track of the slope of the lines, and B -bits are required to save the inverse indicator for partitions. The inverse indicator bits denote whether the data is stored in its actual or inverted form.

5 SIMULATION RESULTS AND WORKLOAD CHARACTERIZATION

5.1 Monte-Carlo Simulation for Memory Lifetime Analysis

We begin our evaluation using Monte-Carlo simulation, making a few simplifying assumptions to perform lifetime simulation in a reasonable amount of time. Much of the prior work discussed in

Fig. 6. The format of the metadata for Aegis (17×31) with block cooperation.

Fig. 7. Lifetimes using standard ECP6, single-level metadata sharing, and multiple-level metadata sharing, with CoV values of (a) 0.20 and (b) 0.30.

this article used Monte Carlo simulation to evaluate their error recovery schemes [4, 8, 16, 32, 38, 41, 42, 46, 47].

5.1.1 Experimental Setup. To evaluate the effectiveness of block cooperation, we exploit Monte Carlo simulation. We model a PCM with 2,048 4K-pages, with 64 memory blocks per page. The memory lifetime is assumed to follow a normal distribution, with a mean of 10^8 . Coefficient of variation (CoV) values of 0.2, and 0.3 are used in our experiments to model different degrees of imperfection in process technology [41, 46, 51]. The higher the CoV, the higher the variability of the lifetimes across memory cells. For each 512 bits of data, 64 bits of metadata is maintained, as is used in standard ECC-based DIMM memories (see Figure 2). We also consider the impact of wear-out on the metadata in our simulations.

The probability of a bit flip in a cell is assumed to be 0.5 throughout the simulations. Similarly to the previous work [16, 32, 41, 42, 46], we assume perfect wear leveling across the memory blocks. The entire page is disabled if ECC is not able to correct faults in a memory block, and the write in that block is uniformly distributed among other blocks.

5.1.2 Lifetime Results Evaluation. The memory lifetimes, when single and multi-level metadata sharing are integrated with ECP, are presented in Figure 7. For single-level metadata, as the CoV parameter is varied from 0.2 to 0.3, lifetime improves over standard ECP6 from 9% to 28%. Using multi-level sharing boosts the lifetime improvement to 12% and 37%, with CoV values of 0.2 and 0.3,

Fig. 8. Lifetimes using standard Aegis, Aegis with single sharing level, and Aegis with multiple sharing level, with CoV values of (a) 0.20 and (b) 0.30.

respectively. As Figure 7 shows, higher CoV results in better lifetime improvements when using metadata sharing (differences between ECP6 and sharing increase from left to right). Therefore, when using technology with high process variation, exploiting a metadata sharing technique with ECP6 is more effective and will increase lifetimes considerably.

The policy for selecting a block for join operations is to find a block with no or the minimum number of faulty cells within the page. We also consider no limitation on the number of blocks for multiple cooperation.

Figure 8 shows page lifetimes with single and multiple levels of sharing integrated with Aegis and compares against Aegis without any cooperative sharing. CoV values of 0.20 and 0.30 are again considered. For single sharing, the maximum retry threshold is set to 4, while this parameter is increased to 8 for multi-sharing. We choose to increase this limit so that we can explore the limits of cooperation in our evaluation. The policy for selecting a block for join operations is random, as there is no notation of the number of faults in Aegis metadata as compared to ECP. The memory lifetime increases by more than 3% (6%) and 8% (14%) when single (multiple) sharing is used. Just as when we added block cooperation to ECP, Aegis also shows better results with higher CoV.

5.1.3 Comparison to Other Error Correction Schemes. To compare the effectiveness of our block cooperation scheme against other rivals approaches, we modeled four other error recovery schemes from specific categories (as explained in Section 2.3).

- PAYG [38]³: This schemes is in category 1 of the error correction schemes. The underlying logic is similar to the ECP scheme, except that the ECP entries are not distributed uniformly. One local, dedicated, and low-latency ECP entry is allocated for each data block. To correct more bit failures, PAYG maintains additional ECP entries in a global shared structure, with higher latency, to reduce the storage overhead. The main advantage of PAYG is that it uses significantly less storage overhead as compared to ECP. However, it requires non-trivial modifications to the PCM memory array and standard DIMM structure.

³We modeled the default PAYG configuration in our simulations.

Fig. 9. The comparative endurance of error correction schemes, with CoV values of (a) 0.20 and (b) 0.30.

- Layered ECP [41]⁴: This scheme lies in the category 1 of the error correction schemes. Similarly to PAYG, Layered ECP uses ECP as underlying method for error correction. However, Layered ECP adds another row, using larger entries to correct errors throughout a 4KB page. With similar added overhead, Layered ECP can provide better endurance as compared to ECP. However, Layered ECP is more complex to implement [41] and requires operating system modifications to allocate an extra row for each page.
- SAFER32 [42]. This scheme is from category 2 of the error correction schemes. SAFER partitions memory blocks into 32 partitions, and uses data inversion for each partition to mask errors. Similarly to ECP [41], the error correction codes are maintained inside of the NVM-DIMM, so no modifications are required of the standard interface.
- Dynamically Replicated Memory (DRM) [19]. This scheme is in category 3 of the error correction schemes. DRM assigns a parity bit to every 8-bit memory cell to detect errors. The first failed bit within a 4KB page causes the block to die, but the page is paired with another spare page to ensure data correctness. So no modifications are needed in the NVM-DIMM to support DRM. However, replicating the data in both pages can rapidly degrade the effective capacity of the memory system and lead to performance loss.

Figure 9 shows the fraction of pages that survive a given number of page writes, while increasing the coefficient of variation from 0.2 to 0.3, for different error correction schemes. DRM is less beneficial as compared to the other error recovery schemes. In addition, memory capacity is reduced to half quickly, since DRM only relies on page pairing just after the first cell failure occurs in the memory. PAYG and SAFER32 perform similar to each other, particularly when the cell lifetime variance is higher. Layered ECP is capable of outperforming PAYG and SAFER32 by 10% in higher CoV, balancing tolerated errors in rows and within a page. Exploiting block cooperation helps ECP6(multiple) and Aegis(multiple) tolerate more than 230 million and 675 million more writes per page as compared to Layered ECP, which is equivalent to a 16% and 29% longer lifetime, respectively, as compared to PAYG.

⁴Our implementation of Layered ECP corrects up to 5 errors in a block, and 35 extra errors per page, with similar storage overhead as compared to ECP.

Table 1. The System Specifications of Our Simulated System

Processor	
ISA	X86
CMP	four-core, out-of-order, 2.66GHz
Interconnect Network	Bus
On-chip Cache Hierarchy	
L1 I-Cache	32KB/64B/8 way, private
L1 D-Cache	32KB/64B/8 way, private
L2 Cache	256KB/64B/8 way, private
L3 Cache	8MB/64B/16 way, shared
Store Policy	Write-Back
Main Memory Configuration	
Total Capacity	8GB (4KB pages, and 64B rows)
Configuration	4 memory controller 1 rank/DIMM, 8 devices/rank, 1 chip for ECC, ECP [41] and Aegis [16] error corrections

5.2 Trace-Driven Simulation for Memory Lifetime Analysis

We use SniperSim [7] to execute workloads and generate memory traces of applications from the SPEC2006 benchmark suite [13]. Table 1 describes the parameters used in the modeled system. Our baseline is a four-core out-of-order multiprocessor that includes three levels of on-chip cache in the memory hierarchy. The benchmarks are executed in rate mode [34], where all the four cores execute the same benchmark.

Memory write requests are captured at the PCM memory and on evictions from the last-level cache (i.e., L3 in our simulations). The granularity of read and write requests are 64 bytes (i.e., one cache line) and are serviced by one of the PCM banks. The first two billion instructions for each benchmark are used for warming up on-chip caches. All references to main memory, each 64 bytes of data, are captured. We terminate trace recording after either 10 million main memory references, or 20 billion executed instructions, whichever occurs first. Note that 10 million references typically correspond to several billions of instruction executed for each benchmark.

5.2.1 NVM Workload Characterization. One of the most valuable insights derived from workload characterization is the identification of key architectural features that will dominate program execution. Past workload characterization has focused on performance optimization [6, 12, 18]. However, in this article, we identifying quantitative metrics for a workload that will most impact non-volatile memory. By providing a set of quantitative workload metrics, we can better understand the relationship between workload and memory lifetimes. We perform characterization using detailed trace-driven simulation that models actual data values transferred between on-chip memory and main memory. In Figure 10, we present an overview of the various steps comprising our framework, which are used for endurance analysis and associated workload characterization.

We study 15 different benchmarks, providing us with a range of different write intensities and behaviors. The percent of bit flips per block on a writeback varies from 5% to 45% across the benchmark suite.⁵ For all of the benchmarks, we used the reference inputs. It is noteworthy that the input of a benchmark can change the size of the memory footprint and even the read/write characteristics of an application. However, as our main goal is to investigate the relationship between

⁵Writing 100 bits to a memory block is approximately equal to 100/512≈20%.

Fig. 10. Simulation framework and methodology for workload characterization and endurance analysis. The * symbol indicates the outputs of our simulation framework.

Table 2. Collected Bit Flip Metrics, Reported at a Block and Page Level, for the Benchmarks

Benchmark	Avg. % of Bit Flip (Block)	Avg. Kilo Bit Flip (Page)	Block-level Bit Flip Intensity	Page-level Bit Flip Intensity
astar	5.7	1.3	577	22.5
calculix	43.0	23.6	1419	85.7
gobmk	13.7	26.0	1439	81.6
gromacs	26.3	45.6	2849	154.9
hmmer	28.1	51.7	2542	155.3
lbm	4.9	1.2	280	17.0
leslie3d	28.1	22.9	1171	69.7
libquantum	4.4	10.8	262	16.0
mcf	8.5	2.4	1947	119.0
milc	20.4	5.8	1454	75.1
omnetpp	11.8	4.8	749	20.9
perlbench	22.7	7.3	953	44.2
sjeng	9.1	4.4	602	23.8
soplex	15.9	8.5	943	58.5
xalancbmk	16.0	6.4	1339	63.9

memory writeback characteristics and memory lifetime, we did not evaluate program behavior with multiple inputs.

The following metrics were captured across all the benchmarks. Results are provided in Table 2 and Table 3:

- *Average % of Bit Flips per Writeback to Blocks.* Applications only modify a limited portion of the 64-byte data block on each memory writeback. As a PCM chip only writes the differences using the DCW mechanism, a limited number of bit flips will be exposed. The higher the number of bit flips, the more lifetime degradation experienced in the PCM. lbm, astar, and

Table 3. Read/Write Access Profiles of Different Benchmarks

Benchmark	#Writes (Million)	#Reads (Million)	#Written Page (Thousand)	Avg. Writes per Page	Spare Factor
astar	4.09	5.91	72.49	56.4	27.6
calculix	2.77	7.23	23.50	117.8	85.1
gobmk	4.83	5.17	6.33	763.1	316.0
gromacs	0.44	0.83	1.29	338.6	1552.8
hammer	2.79	2.85	7.78	358.8	257.0
lbm	3.35	6.65	52.36	64.0	38.2
leslie3d	2.57	7.43	13.52	190.1	147.9
libquantum	3.44	6.56	8.20	419.9	243.9
mcf	1.70	8.30	5.31	320.15	376.65
milc	3.80	6.20	108.5	35.02	18.43
omnetpp	3.77	6.23	41.57	90.8	48.1
perlbench	2.08	3.91	30.97	67.3	64.6
sjeng	4.47	4.53	44.03	101.6	45.4
soplex	3.69	6.31	22.43	164.6	89.2
xalancbmk	1.98	8.02	25.56	77.4	78.3

libquantum benchmarks only experience, on average, around 5% bit flips on each writeback, while for the calculix benchmark, the percentage grows to 43%. The average percentages of bit flips per writeback across the studied benchmarks is 17.7%.

- *Average Kilo Bit Flip for Pages.* This metric captures the number of bit flips at a page-level granularity. hammer and lbm exhibit 51.7- and 1.2-Kilo bit flips per page, on average, which are the two ends of spectrum, respectively. The average for this metric is 16.5-Kilo bit flips across all benchmarks.
- *Block-level Bit Flip Intensity.* If one of the memory blocks within a page experiences more errors than the tolerable limit, then the operating system will discard entire the page. Therefore, a single memory block of the 64 blocks in a page can dictate the entire page's lifetime. In other words, approximately $1/64 \approx 2\%$ of the blocks that experience the highest number of bit flips contribute the most to page wear-out and reduce memory lifetime. The Block-level Bit Flip Intensity metric is computed as the average number of bit flips per block for the 2% of the blocks that experience the highest number of bit flips. A well-designed wear-leveling algorithm should effectively distribute these bit flips across all the blocks in memory.
- *Page-level Bit Flip Intensity.* In case of a page failure, the operating system needs to remap the page to another location in memory. This gradually reduces the spare blocks available and shrinks the size of the memory system. Consequently, the pressure the on memory system escalates over time and memory lifetime decreases. The Page-level Flip Intensity metric is computed as the average number of Kilo bit flips (i.e., 1,000 bit flips) in a page, for the 2% of pages that experience the highest number of bit flips. We expect that, in the future, these pages will cause more wear-out.
- *Number of Writes (Reads).* This metric captures the number of write (read) accesses that are captured when recording the trace of each benchmark (a trace is either 10M references to main memory, or 20B instructions, whichever comes first).
- *Written Pages.* This metric records the number of pages that experienced at least one block write.

Fig. 11. Distribution of (a) writes and (b) bit flips at block level for different benchmarks. The red dot indicates the median value of the distribution.

- *Average Writes per Page*. This metric reports the average number of writes to each page.
- *Spare Factor*. Having a larger capacity memory system, while running an application, provides more spare pages that can come to the rescue during wear-leveling and error correction to prolong memory lifetimes. The spare factor includes the capacity of the whole PCM DIMM over the capacity of written pages (i.e., a value greater than 1, where higher is better).

Although simplified metrics are good indicators to distinguish workload behaviors, the distribution of writes or bit flips in individual benchmarks cannot be easily captured in a single metric. To this end, we use a Violin plot to show the shape of the probability density of writes and bit flips at both a block-level and page-level granularity.

Figure 11(a) and (b) present the number of writes and bit flips encountered, at a block level, across the 13 benchmarks. Wider sections of the Violin plot represent a higher density of the points for the given value, while the skinnier sections represent a lower density.

Some of the benchmarks are less variable than the others. For instance, *astar*, *lbm*, *libquantum*, and *soplex* show much less variation than *xalancbmk*, *perlbench*, and *gromacs*. The number of writes to a single block can reach 972, 177, and 90 for these three benchmarks, respectively.

For *xalancbmk*, the Violin shape is tall and narrow, which means the points are spread across a wide range of values, even though the number of bit flips at a page level can reach 78.1 Kilo bit flips.

Figure 12 shows the distribution of writes and bit flips at a page-level granularity. In contrast to the block-level granularity, the shapes of writes and bit flips are similar in many of the benchmarks. But aggregating behavior at a page level, the similarity decreases. The number of writes per page can reach to 6,422 for *xalancbmk*, while *lbm* only experiences one write per block in some pages (at most 64 writes per page). *xalancbmk* experiences a maximum of 572-Kilo bit flips in a page, which is the highest number of bit flips within a page for the workloads studied. For the same workload, the mean number of bit flips across all the pages is only 6.4 Kilo. *lbm* has a median value of 26 bit flips per page, and a mean of 1.2-Kilo bit flips per page, and has the smallest total number of bit flips.

5.2.2 Experimental Setup. Memory lifetime simulation involves a few challenges, as it is challenging to simulate memory with actual data values to accurately assess memory wear-out for long-running programs. This challenge is further exacerbated when the size of the simulated memory is large [46]. If we adopt Monte Carlo simulation, then we would generally assume that the

Fig. 12. Distribution of (a) writes and (b) bit flips at a page level for different benchmarks. The red dot shows the median value of the distribution.

wear-leveling algorithm is perfect, that all of the blocks in a page experience exactly the same number of writes, and that the probability of a bit flip is 50% across all blocks. However, real applications rarely access all of the blocks in the memory and certainly do not access every data block uniformly.

In trace-driven simulation, applications can touch a large number of pages. Since memory reliability simulation needs to be carried out at the granularity of a bit level, simulating a large number of pages results in a dramatically larger memory footprint for the simulation, reaching many gigabytes in size. To reduce simulation time, while not sacrificing accuracy in our results, we assume the number of writes that each cell can tolerate before a failure follows a normal distribution, with a mean (μ) of 10^5 (versus 10^8 , which is the value used in our Monte Carlo simulations). The number of simulated memory pages is assumed equal to the memory footprint of an application. However, we scaled the final results with respect to the total size of the main memory to make results across benchmarks comparable and meaningful. Note that the former optimizations may introduce small artifacts into the results but greatly reduce our simulation time.

We replay the trace of applications repeatedly, tracking the accesses made to each memory block. Moreover, we keep track of modifications at a bit-level granularity to check whether the number of writes to a bit address exceeds wear-out limits or not. Logical-to-physical address mapping is performed, along with applying a Start-Gap wear-level mechanism [39]. If a selected error recovery scheme is not capable of tolerating faults in a given data block, then the whole page is dead and is marked unusable by the operating system. Our framework maps the address of the dead page to another available physical page. The simulation continues until half of the physical pages in our simulation are dead.

5.2.3 Lifetime Analysis. For lifetime analysis, we compare the volume of writes that occur before half of the availability memory pages die across the four error recovery schemes: ECP6, ECP6 with multiple block cooperation, Aegis 17×31 , and Aegis 17×31 with multiple block cooperation. We measure write volume in petabytes and use this metric to evaluate memory durability for each benchmark. This metric helps to identify common memory access behaviors across workloads that impact memory durability and allows us to compare and evaluate different lifetime enhancement strategies with block cooperation. Figure 13 and Figure 14 show the number of petabyte writes, across different error recovery schemes, and with a CoV of 0.2 and 0.3, respectively.

Block cooperation significantly increases memory durability in perlbench for both ECP and Aegis schemes. For instance, we see an increase of 60% (68%) and 18% (30%) in terms of longer

Fig. 13. The Y-axis (petabyte writes) shows the number of writes possible before half of the memory capacity becomes unavailable. The CoV is equal to 0.2.

Fig. 14. The Y-axis (petabyte writes) shows the number of writes possible before half of the memory capacity become unavailable. The CoV is equal to 0.3.

lifetimes achieved, with a CoV of 0.2 (0.3) for ECP and Aegis, respectively. However, the average improvement over all the benchmarks is 18% (24%) and 9% (19%) when block cooperation is used with ECP and Aegis, with a CoV of 0.2 (0.3), respectively.

As Figure 13 and Figure 14 show, we see that memory durability is impacted more in *gromacs*, *calculix*, *hammer*, *leslie3d*, and *xalanbm* than in the other benchmarks. These workloads generally have higher block-level and page-level bit flip intensities as compared to the other benchmarks. Increasing the CoV from 0.2 to 0.3 reduces the memory lifetime by 31% on average.

lbm achieves the longest memory lifetime among all benchmarks, which is tied to the memory characteristics of this benchmark. In Figure 11 and Figure 12, which show the distribution of writes to blocks and pages, *lbm* has the lowest values. *lbm* has the smallest number of bit flips per page—the average number of bit flips per block writeback is very small (i.e., 4.9%). The number of written pages, and the number of writes, is large. However, only one write is performed to each block in a page for *lbm*. Note that metadata sharing in ECP can only increase lifetime by ~3–4% for *lbm*, while when using data layout reorganization in Aegis, lifetime can increase by 25%.

The correlation between block-level bit flip intensity with the average number of bit flips per block is moderately positive (0.66), while the correlation between page-level bit flip intensity with the average bit flip per page is strongly positive (0.95). The lifetime correlation with bit flips at a block level and page level is moderately negative and varies between ~ -0.44 and -0.54 . Benchmarks with a higher number of pages accessed, such as *lbm*, *astar*, and *omnetpp*, experience longer lifetimes, as there is a greater chance to balance wear-out across more pages. Note that there is moderately strong correlation (almost 0.50) between the number of written pages and the benchmark lifetime.

Fig. 15. Memory durability (as petabyte write) versus the average error per page for different error correction schemes. Using block cooperation increases the number of correctable errors per page and increases metadata utilization for both ECP and Aegis.

Since block cooperation more effectively uses the metadata, it can tolerate more errors, thereby improving memory durability. Figure 15 shows the petabyte writes occurring (as a representative metric of memory durability) for three benchmarks, each possessing a different number of tolerable errors per page. The results are for different error correction schemes with and without block cooperation and vary the CoV. Aegis can correct a larger number of errors and always produces more correctable errors as compared to ECP. Block cooperation increases the number of correctable errors, although its effectiveness varies with respect to the error correction scheme and the specific benchmark considered. Since ECP can only tolerate six errors, the metadata are always underutilized. Multi-block cooperation shows significant potential, across all of the benchmarks, in tolerating more errors and increasing memory lifetimes. For omnetpp using Aegis, block cooperation provides a significant improvement, while xalancbmk sees little improvement.

5.3 Block Cooperation Discussion: Complexity, Overhead, and Compatibility

Exploiting metadata sharing for ECP [41], and data layout reorganization for Aegis [16], we can improve memory lifetimes considerably. The block cooperation concept is applicable to all block-level error corrections (i.e., categories 1 and 2) if the standard memory interface (with the ECC chip) is used for error correction. However, minor modifications to a metadata format are required to be tailored to a specific scheme. For instance, the data layout organization proposed for Aegis can be modified to be used for other category 2 schemes, such as SAFER [42] (the modification required is due to the format of metadata, which is different in SAFER). Similarly, the metadata sharing proposed for ECP can be also customized for other category 1 schemes, such as REMAP [46].

However, block cooperation is similar to other error correction schemes [4, 16, 32, 41, 42, 49] in that it adds some complexity to the memory system. Block cooperation schemes are fully compatible with NVM-based DIMMs, so they incur no modifications. Hence, all the required metadata can be stored in the ECC chip. However, the logic for block cooperation presented through the state-chart in Figure 3 is required to implement in the memory controller to support block cooperation.

It might seem that accessing cooperative blocks introduces some performance degradation, as single or multiple accesses are required for error recovery. Nonetheless, the row buffer in the main memory architecture *overfetches* data, e.g., 4KB is read for a 64B request. This will impact the latency associated with subsequent memory accesses within the page (i.e., a row buffer hit is around 20 ns [21]). In the absence of block cooperation, the entire page should be disabled, which then would lead to an access to the next level of the memory hierarchy or a page swap by the operating system. Either of these approaches results in a high-latency access (on the order of

thousands of cycles). Therefore, saving blocks from failure, at the added cost of a small latency, is much more palatable, especially versus disabling the page.

Page-level paring schemes, such as DRM [19] and Zombie [4], can be used as a second layer of defense, with block cooperation, to further increase memory lifetimes. However, they require access to an additional page of memory, which is costly in terms of performance and energy consumption, as compared to block cooperation, which tries to increase correctable errors by exploiting resources within a page. Limiting block cooperation to within a page's address range and applying this change at a block level increases row buffer hits and incurs negligible power increases and performance overheads. Supporting block cooperation across page boundaries incurs design complexity, energy, and performance overhead for error correction. Additionally, having cooperative blocks maintain only within page boundaries provides better utilization of metadata for error correction in a multiple block scenario. Moreover, as all the required metadata for error recovery is confined within a page, the block cooperation scheme is completely compatible with page-level wear-leveling schemes [14, 51] and hence does not need any modification to the underlying logic of the error recovery scheme.

In block-level wear-leveling schemes such as the Start-Gap mechanism [39], block movements or rotations are used to distribute writes uniformly across blocks. The metadata associated with data blocks need to be updated on every write, independent of whether the write operation is due the application performing a write-back or is invoked by the wear-leveling mechanism. In the presence of the block cooperation, a block-level wear-leveling mechanism may exchange the contents of a *Co-op* block with several stuck-at faults with another block (i.e., the *Gap* block in Start-Gap mechanism). As the metadata of a *Co-op* block is distributed across other blocks, the metadata of the associated blocks should be updated accordingly. This procedure is the same as when data are written back to a *Co-op* block from the on-chip memory hierarchy.

In ECP, when using metadata sharing, read/write operations from/to only the *Co-op* blocks impose extra read/write operations from/to metadata, while the accesses to the *shared* and *shared+* blocks are performed with no extra latency and further memory access. In Aegis, when using data layout reorganization, every read from the cooperative blocks requires reading data and metadata from other blocks. For writing to the cooperative blocks, reading data from the other blocks is required, involving further metadata updates. Therefore, error recovery can take more cycles in Aegis as compare to ECP with block cooperation. The worst-case scenario (from a performance perspective) associated with block cooperation is when an access to one block leads to a chain of accesses to all the other blocks within a page to correct the errors. The performance loss in the worst case is bounded; this is a rare case, considering that errors follow a normal distribution in the memory cells. Nonetheless, the performance overhead is negligible over the lifetime of the system. Performance only gradually degrades as more errors occur in the memory system.

6 CONCLUSION

Limited write endurance is the *primary* shortcoming of resistive memories. Hence, error recovery schemes are required to correct errors and boost memory lifetimes of resistive memory devices. Once an error recovery scheme fails to recover from faults in a data block, the entire physical page associated with that block is disabled and becomes unavailable to the physical address space. To reduce page waste caused by early block failures, and increase utilization of metadata, we proposed block cooperation. Our approach allows live blocks to help failed blocks to remain alive and extend a page's lifetime.

We combined the proposed technique with two state-of-the-art error recovery schemes, ECP and Aegis. The goal was to increase memory lifetimes, with only minor modifications to the underlying error correction mechanism. We evaluated our propose modifications to ECP (metadata sharing)

and Aegis (data layout reorganization). We used both Monte Carlo and trace-driven simulation to evaluate our approach. We also presented a workload characterization scheme that focuses on memory lifetimes, providing insight on how workload behavior is tied to the success of different error recovery schemes.

Employing block cooperation on top of Error Correction Pointer (ECP) and Aegis increased memory lifetimes by 37%, and 14% on average, respectively. Lifetimes can be increased further by 60% (68%) by exploiting metadata sharing, or through data layout reorganization 13% (30%), considering CoV of 0.2 (0.3), respectively.

REFERENCES

- [1] Hoda Aghaei Khouzani, Yuan Xue, Chengmo Yang, and Archana Pandurangi. 2014. Prolonging PCM lifetime through energy-efficient, segment-aware, and wear-resistant page allocation. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*. ACM, 327–330.
- [2] Aravinthan Athmanathan, Milos Stanisavljevic, Nikolaos Papandreou, Haralampos Pozidis, and Evangelos Eleftheriou. 2016. Multilevel-cell phase-change memory: A viable technology. *IEEE J. Emerg. Select. Top. Circ. Syst.* 6, 1 (2016), 87–100.
- [3] Manu Awasthi, Manjunath Shevgoor, Kshitij Sudan, Bipin Rajendran, Rajeev Balasubramonian, and Viii Srinivasan. 2012. Efficient scrub mechanisms for error-prone emerging memories. In *Proceedings of the 2012 IEEE 18th International Symposium on High Performance Computer Architecture (HPCA'12)*. IEEE, 1–12.
- [4] 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 *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 452–463.
- [5] Seungcheol Baek, Hyung Gyu Lee, Chrysostomos Nicopoulos, and Jongman Kim. 2012. A dual-phase compression mechanism for hybrid DRAM/PCM main memory architectures. In *Proceedings of the Great Lakes Symposium on VLSI*. ACM, 345–350.
- [6] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 72–81.
- [7] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. IEEE, 1–12.
- [8] Jie Chen, Guru Venkataramani, and H. Howie Huang. 2014. Exploring dynamic redundancy to resuscitate faulty PCM blocks. *ACM J. Emerg. Technol. Comput. Syst.* 10, 4 (2014), 31.
- [9] Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. IEEE, 347–357.
- [10] Woo-Yeong Cho, Kwang-Jin Lee, and Hye-jin Kim. 2009. Phase change memory devices and systems, and related programming methods. (May 5 2009). Patent No. 7,529,124.
- [11] Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, et al. 2012. A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth. In *Proceedings of the 2012 IEEE International Solid-State Circuits Conference*. IEEE, 46–48.
- [12] Thomas M. Conte and W.-M. W. Hwu. 1991. Benchmark characterization. *Computer* 24, 1 (1991), 48–56.
- [13] SPEC CPU2006. 2006. Standard Performance Evaluation Corporation. Retrieved from <https://www.spec.org/cpu2006/>.
- [14] G. Dhiman, R. Ayoub, and T. Rosing. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the Design Automation Conference (DAC'09)*. 664–669.
- [15] Ali Eslami, Alfredo Velasco, Alireza Vahid, Georgios Mappouras, Robert Calderbank, and Daniel J. Sorin. 2015. Writing without disturb on phase change memories by integrating coding and layout design. In *Proceedings of the 2015 International Symposium on Memory Systems*. ACM, 71–77.
- [16] Jie Fan, Song Jiang, Jiwu Shu, Youhui Zhang, and Weimin Zhen. 2013. Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 433–444.
- [17] David Harel. 1987. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.* 8, 3 (1987), 231–274.
- [18] Kenneth Hoste and Lieven Eeckhout. 2007. Microarchitecture-independent workload characterization. *IEEE Micro* 27, 3 (2007).

- [19] Engin Ipek, Jeremy Condit, Edmund B. Nightingale, Doug Burger, and Thomas Moscibroda. 2010. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *ACM SIGARCH Computer Architecture News*, Vol. 38. ACM, 3–14.
- [20] ITRS. 2015. More Moore. Retrieved from <http://www.itrs2.net/>.
- [21] Bruce Jacob, Spencer Ng, and David Wang. 2010. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann.
- [22] Amin Jadidi, Mohammad Arjomand, Mohammad Khavari Tavana, David R. Kaeli, Mahmut T. Kandemir, and Chita R. Das. 2017. Exploring the potential for collaborative data compression and hard-error tolerance in PCM memories. In *Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*. IEEE, 85–96.
- [23] Lei Jiang, Youtao Zhang, and Jun Yang. 2011. Enhancing phase change memory lifetime through fine-grained current regulation and voltage upscaling. In *Proceedings of the 2011 International Symposium on Low Power Electronics and Design (ISLPED'11)*. IEEE, 127–132.
- [24] Lei Jiang, Youtao Zhang, and Jun Yang. 2014. Mitigating write disturbance in super-dense phase change memories. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. IEEE, 216–227.
- [25] Madhura Joshi, Wangyuan Zhang, and Tao Li. 2011. Mercury: A fast and energy-efficient multi-level cell based phase change memory system. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 345–356.
- [26] Kinam Kim. 2005. Technology for sub-50nm DRAM and NAND flash manufacturing. In *Proceeding of the 2005 IEEE 51st International Electron Devices Meeting, Technical Digest*. IEEE, 323–326. DOI : [10.1109/IEDM.2005.1609340](https://doi.org/10.1109/IEDM.2005.1609340)
- [27] Kinam Kim and Su Jin Ahn. 2005. Reliability investigations for manufacturable high density PRAM. In *Proceedings of the 43rd Annual 2005 IEEE International Reliability Physics Symposium*. IEEE, 157–162.
- [28] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 2–13.
- [29] Suyoun Lee, Jeung-hyun Jeong, Taek Sung Lee, Won Mok Kim, and Byung-ki Cheong. 2009. A study on the failure mechanism of a phase-change memory in write/erase cycling. *IEEE Electron Device Lett.* 30, 5 (2009), 448–450.
- [30] Rakan Maddah, Rami Melhem, and Sangyeun Cho. 2015. Rdis: Tolerating many stuck-at faults in resistive memory. *IEEE Trans. Comput.* 64, 3 (2015), 847–861.
- [31] Rakan Maddah, Seyed Mohammad Seyedzadeh, and Rami Melhem. 2015. CAFO: Cost aware flip optimization for asymmetric memories. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 320–330.
- [32] 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 *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12)*. IEEE, 1–12.
- [33] Azalia Mirhoseini, Miodrag Potkonjak, and Farinaz Koushanfar. 2015. Phase change memory write cost minimization by data encoding. *IEEE J. Emerg. Select. Top. Circ. Syst.* 5, 1 (2015), 51–63.
- [34] Prashant J. Nair, Chiachen Chou, Bipin Rajendran, and Moinuddin K. Qureshi. 2015. Reducing read latency of phase change memory via early read and Turbo Read. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 309–319.
- [35] Poovaiah M. Palangappa and Kartik Mohanram. 2015. Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories. In *Proceedings of the 25th Great Lakes Symposium on VLSI*. ACM, 221–224.
- [36] P. M. Palangappa and K. Mohanram. 2017. CompEx++: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVMs. *ACM Trans. Arch. Code Optimiz.* 14, 1 (2017), 10.
- [37] N. Papandreou, H. Pozidis, T. Mittelholzer, G. F. Close, M. Breitwisch, C. Lam, and E. Eleftheriou. 2011. Drift-tolerant multilevel phase-change memory. In *Proceedings of the 2011 3rd IEEE International Memory Workshop (IMW'11)*. IEEE, 1–4.
- [38] Moinuddin K. Qureshi. 2011. Pay-as-you-go: Low-overhead hard-error correction for phase change memories. In *Proceedings of the 44th Annual International Symposium on Microarchitecture*. ACM, 318–328.
- [39] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd Annual Symposium on Microarchitecture*. ACM, 14–23.
- [40] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 24–33.
- [41] Stuart Schechter, Gabriel H. Loh, Karin Strauss, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *ACM SIGARCH Computer Architecture News*, Vol. 38. ACM, 141–152.
- [42] Nak Hee Seong, Dong Hyuk Woo, Vijayalakshmi Srinivasan, Jude A. Rivers, and Hsien-Hsin S. Lee. 2010. SAFER: Stuck-at-fault error recovery for memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 115–124.

- [43] Seyed Mohammad Seyedzadeh, Rakan Maddah, Donald Kline, Alex K. Jones, and Rami Melhem. 2016. Improving bit flip reduction for biased and random data. *IEEE Trans. Comput.* 65, 11 (2016), 3345–3356.
- [44] Mohammad Khavari Tavana, Yungsi Fei, and David Kaeli. 2018. Nacre: Durable, secure and energy-efficient non-volatile memory utilizing data versioning. *IEEE Trans. Emerg. Top. Comput.* 99 (2018), 1–1. DOI: <https://doi.org/10.1109/TETC.2017.2787622> (In press).
- [45] Mohammad Khavari Tavana and David Kaeli. 2017. Cost-effective write disturbance mitigation techniques for advancing PCM density. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press.
- [46] Mohammad Khavari Tavana, Amir Kavayan Ziabari, Mohammad Arjomand, Mahmut Kandemir, Chita Das, and David Kaeli. 2017. REMAP: A reliability/endurance mechanism for advancing PCM. In *Proceedings of the International Symposium on Memory Systems (MEMSYS'17)*. ACM, New York, NY, 385–398. DOI: <https://doi.org/10.1145/3132402.3132421>
- [47] Mohammad Khavari Tavana, Amir Kavayan Ziabari, and David Kaeli. 2017. Live together or die alone: Block cooperation to extend lifetime of resistive memories. In *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 1098–1103.
- [48] Rujia Wang, Lei Jiang, Youtao Zhang, and Jun Yang. 2015. SD-PCM: Constructing reliable super dense phase change memory under write disturbance. *ACM SIGPLAN Not.* 50, 4 (2015), 19–31.
- [49] 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 *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 466–477.
- [50] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. DEUCE: Write-efficient encryption for non-volatile memories. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 33–44.
- [51] Wangyuan Zhang and Tao Li. 2009. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2–13.
- [52] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 14–23.

Received March 2018; revised June 2018; accepted July 2018