Live Together or Die Alone: Block Cooperation to Extend Lifetime of Resistive Memories

Mohammad Khavari Tavana, Amir Kavyan Ziabari, David Kaeli
Department of Electrical and Computer Engineering
Northeastern University, Boston (MA), USA
Email: {mkhavaritavana, aziabari, kaeli}@ece.neu.edu

***Abstract*—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. Employing block cooperation at a single level (or multiple levels) on top of ECP and Aegis, we can increase memory lifetimes by 28% (37%), and 8% (14%) on average, respectively.

I. Introduction

Due to shrinking cell sizes in successive technology nodes, DRAM scaling has become a challenging problem [1]. Due to limits on the electrical charge that can be trapped in the DRAM capacitors, retention time is reduced, dramatically increasing the chance of unreliable sensing [1], [2]. Resistive memories have been proposed as a viable alternative [3]. Instead of representing the bit state as the 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 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 and the ability to retain cell state for more than ten years [1]. These features pave the way for PCM to be mass produced and commercialized [4], [5]. Furthermore, given that recent devices support multiple bits per cell in PCM [5], this can double or triple the memory density.

Despite these desirable features, write endurance still looms as the major drawback of using PCM, and currently prevents PCM from replacing current main memory technology [1].

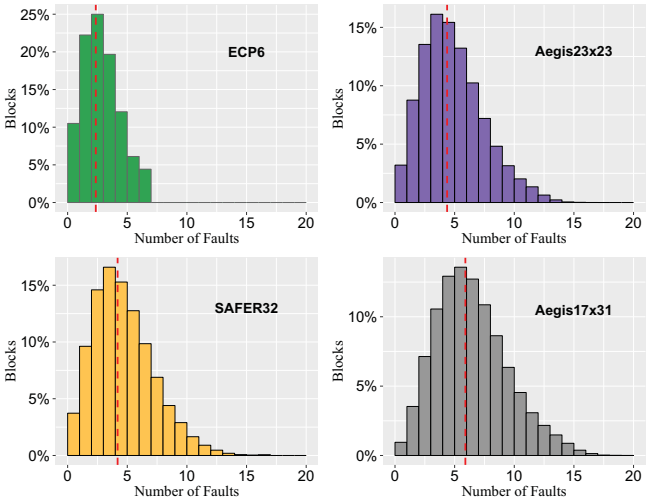


Fig. 1. The percentage of blocks that experience faults when all the pages in the memory die. Memory is protected by ECP6, Safer32, Aegis 23×23, and Aegis 17×31 error recovery schemes. The dotted line represents the average number of faults tolerated over all the data blocks.

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) PCM can tolerate 10^6 write cycles [6], [7]. The write cycle endurance for PCM is expected to further deteriorate as the cell geometries are reduced in future technologies [1].

To mitigate the write endurance problem, proactive approaches can be introduced to reduce the number of writes or bit flips in the PCM. When a fault is detected in PCM, an error correction mechanism is required for recovery. Extra memory bits are used as the metadata for the Error-Correcting Code (ECC) scheme to detect and correct the faulty bits, increasing memory lifetimes. Since error profiles in resistive memories are handled completely differently than in DRAM, a number of novel error correction schemes have been proposed to recover from stuck-at faults in resistive memories [7]–[13].

We model more than two thousand 4K pages, each containing 64 memory blocks. We continuously write to this memory and record the number of errors in each data block when its associated page is taken offline (we describe our

simulation methodology in detail in Section IV-A). Once an error correction scheme can no longer tolerate faults in a data block, the entire physical page associated with that data block needs to be disabled and becomes unavailable to the physical address space. In Fig. 1 we plot the percentage of blocks (y-axis) that experience a specific number of faults (x-axis) at the point in time when all the pages in the memory fail. We present results for ECP6 [7], Safer32 [8], and Aegis [9] (with two different error recovery configurations). The dashed line represents the average number of errors tolerated over all the data blocks. Even though ECP6 can tolerate up to six faults per data block, almost 77% of data blocks experience three or fewer faults. On the other hand, Aegis 17×31 can tolerate 10 faults, and is able to tolerate more faults (probabilistically). However, almost 50% of all of the data blocks tolerate five or fewer faults. Our motivating example shows that ECC-related metadata is not utilized efficiently, or possibly that the faults are distributed unevenly in these schemes.

Motivated 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 performance and increase memory lifetime.

To evaluate the effectiveness of our proposed approach, we exploit block cooperation on top of ECP [7], and Aegis [9] error correction schemes. Note that both Aegis [9] and SAFER [8] are based on partitioning and bit inversion to mask errors, therefore, the proposed scheme for Aegis is also applicable for the SAFER scheme.

Block cooperation is realized through metadata sharing in ECP. Metadata sharing can be performed in single level, where one data block shares its unused metadata with another data block, or for 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 in order 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.

The rest of the paper is organized as follows: in Section II we briefly present related work, reviewing error detection and failure mechanisms in PCM. We present our block cooperation scheme in Section III. Metadata sharing for ECP, and data layout reorganization for Aegis, are covered in Section IV, and Section VI concludes the paper.

II. Background and Related Work

To mitigate the write endurance problem in PCMs, prior work proposed techniques to reduce the write traffic using write buffers [3], compression algorithms [14], and encoding data [11]. Although these techniques are effective in prolonging memory lifetime, error correction mechanisms still need to be included to handle faults due to wear-out.

The error profile in PCM devices is completely different from the profile of DRAM errors. In DRAM, where the errors tend to be soft errors and occur infrequently, PCM errors are permanent and accumulate over time. PCM faulty cells are stuck at either '0' or '1'. The contents of the faulty cells cannot be changed, but they are still readable. Therefore, faults can be detected on write accesses using read after write verification. Whenever an error is detected in a block, the ECC can be used to correct the error. Read verification logic is already implemented in MLC memories and is used for error detection in our work (as considered in prior studies [7]–[13]).

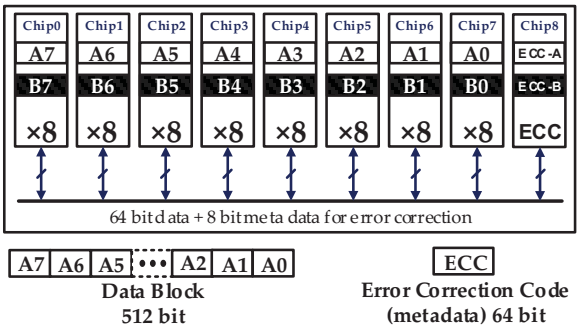


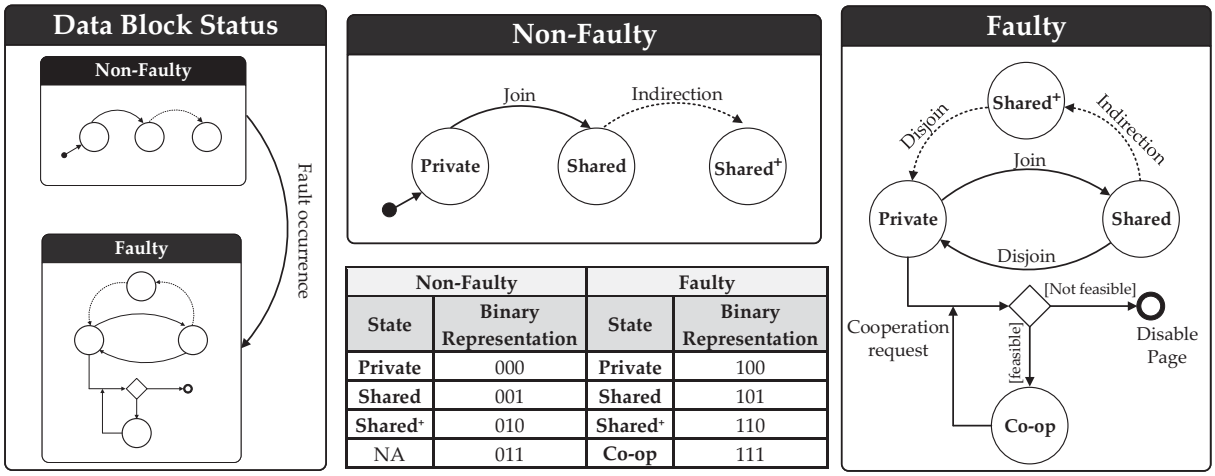
Fig. 2. Standard ECC-based dual in-line memory module (DIMM) memory interface.

A number of ECC-based schemes have been proposed for resistive memories. The Error Correction Pointer (ECP) [7] and PAYG [12] schemes are based on locating faulty bits and replacing them with non-faulty ones. In Free-p [10], when ECC is no longer effective to save a block, a pointer to a spare block is replicated multiple times in the faulty data block. Free-p employs redundant blocks, substituting for the faulty blocks. Zombie [11] prolongs memory lifetime by reusing non-faulty blocks of disabled pages. Aegis [9], SAFER [8], and RDIS [13] mask errors by storing data in their inverted form. In contrast to ECP, these schemes are more effective at boosting memory lifetime, since they can tolerate a larger number of faults. However, they impose more complexity to the memory controller. The cooperative block proposed in this paper is orthogonal to the ECC mechanism used, and increases ECC's ability to tolerate more faults per data block.

We assume we are using standard ECC-based Dual In-Line Memory Module (DIMM) memory, as shown in Fig. 2. The memory chips store data in 64B rows, spread over eight chips. One redundant chip is used to store the ECC, which is treated as metadata. Therefore, for every data block (e.g., Block A or B), 8Bs of storage are required to keep the metadata (e.g., ECC-A or ECC-B). The standard DIMM has a 64-bit data path, therefore eight burst accesses are required to read or write a block of data.

III. Block Cooperation

Applications do not access all data blocks uniformly. The write endurance of cells within a data block is also not equal across all cells. Some will wear out sooner than others (surviving fewer writes). When ECC is no longer able to tolerate faults in a data block, the operating system will remove the entire page that contains that block. Hence, the weakest block in a page dictates the fate of the rest of the block in a page. This is while the metadata of the other blocks within the page remain under-utilized, and while the



Non-Faulty		Faulty	
State	**Binary Representation**	**State**	**Binary Representation**
Private	000	**Private**	100
Shared	001	**Shared**	101
Shared+	010	**Shared+**	110
NA	011	**Co-op**	111

Fig. 3. Block cooperation statechart. At the highest level of the statechart hierarchy, the state of blocks is either *faulty* or *non-faulty*. Each of these superstates contains various states, whichtransitions to a new state based on the specific operation.

other blocks in the page may experience no or only a few faults. Block cooperation techniques address this problem by allowing data blocks to share their data and/or metadata cooperatively, extending memory lifetime.

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 the 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 cooperating block or blocks.

To initiate the cooperation procedure, the memory controller needs to be able to find a candidate block. The selection policy can be done randomly or can follow a set policy. To record each state and handle transitions through these states during the lifetime of a data block, we use *statecharts*. Depending on the state of a data block, the memory controller is responsible for updating the block state and initiating any required operations to handle block cooperation. Statecharts are more flexible than finite-state machines. Fig. 3 shows a statechart that describes our block cooperation algorithm. Note that the transitions presented by dotted arrows only occur during multi-level cooperation. Fig. 3 also shows the binary representation of each state. For every transition, the control field of the metadata is updated by this binary representation.

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 which can resuscitate the failed block, 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.

IV. Block Cooperation Adoption

To show the effectiveness of our approach, we select two error correction schemes designed specifically for resistive memories: i) ECP [7], and ii) Aegis [9]. 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 reorganization* techniques, respectively. Before diving into these details, we describe our experimental methodology next.

A. 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. Note that changing the CoV captures the impact of process variation on the memory cells of the PCM devices, 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 Fig. 2). We also model 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. Perfect wear leveling across the memory blocks is assumed in work, which is in line with previous wear leveling studies for PCMs [15]. The entire page is disabled if ECC is not able to correct faults in a memory block, unless block cooperation is exploited to revive the block.

B. Block Cooperation in ECP

The Error Correction Pointer (ECP) [7] 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. Assuming 8B of metadata are used for a 64B data block, ECP provides the ability to correct up to six faults, which we will refer to as ECP6 throughout the rest of this paper.

In the standard 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 III, 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. Fig. 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. Five bits indicate the number of extra ECP entries required by that block, thereby providing the capability of tolerating 36 faults for a block (in the best case). Sharing requires additional fields in the metadata, which results in reducing ECP entries to 5. In other words, we can only have 6 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.

The policy for selecting a block for join operations involves finding a block with no or the minimum 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, if there is a row buffer hit in the page is small [16].

The memory lifetimes, when single and multi-level metadata sharing are integrated with ECP, are presented in Fig. 5. For single-level metadata, as the CoV parameter is varied from 0.2 to 0.3, lifetime improves over standard ECP6 from

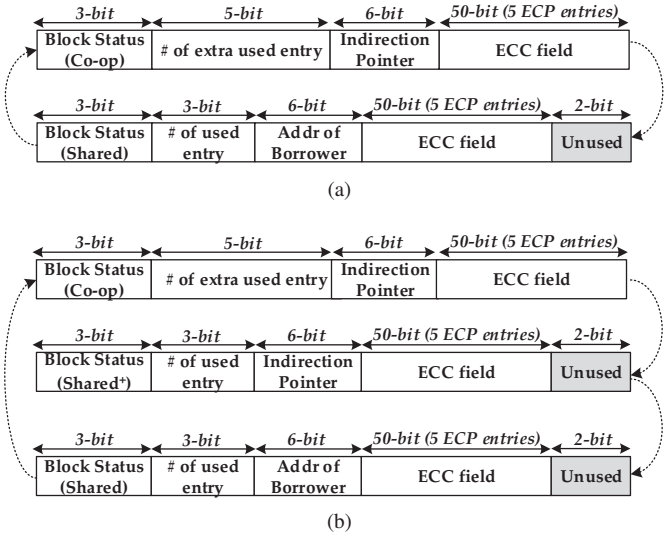


Fig. 4. ECP metadata format when using (a) single level, and (b) multiple level, metadata sharing.

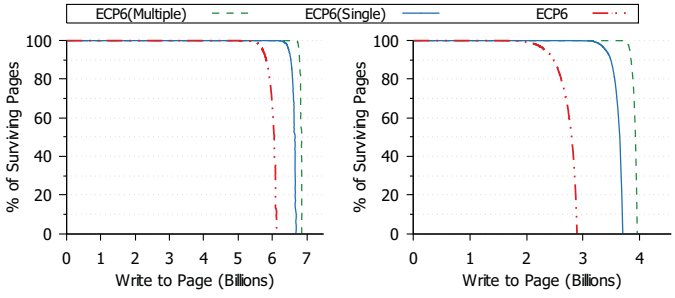


Fig. 5. Lifetimes using standard ECP6, single level metadata sharing, and multiple level metadata sharing, with CoV values of (a) 0.20, and (b) 0.30.

9% to 28%. Using multi-level sharing boosts the lifetime improvement to 12% and 37%, with CoV values of 0.2 and 0.3, respectively. As Fig. 5 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 metadata sharing technique with ECP6 is more effective and will increase the lifetime considerably.

C. Block Cooperation in Aegis

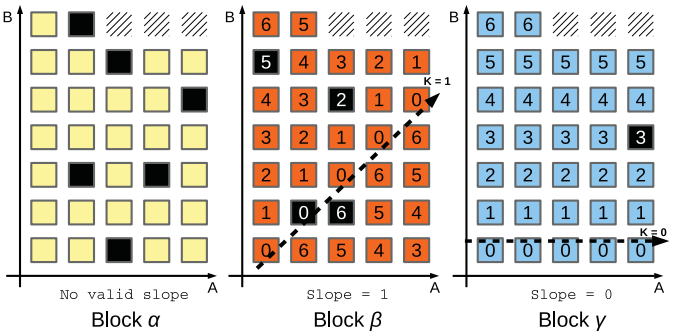
Aegis [9], which similar to SAFER [8], uses partitioning and bit inversion to mask stuck-at errors, storing faulty values in their inverted form. To exploit fault masking in these error correction schemes, faults should be isolated logically using partitioning. The goal of the partitioning scheme is to guarantee the existence of no two faults in the same partition. Aegis performs this partitioning by mapping data blocks to a 2-D Cartesian plane with $A \times B$ elements. The plane for a simplified 32-bit data block in a 5×7 organization is shown in Fig. 6a. In this form, each error is considered as a point on the plane, while partitions are the B parallel lines.

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, error recovery is not possible. In Fig. 6a, three different blocks are shown. No valid slope is found for block α, whereas blocks β and γ can tolerate the fault pattern with a slope one and zero, respectively.

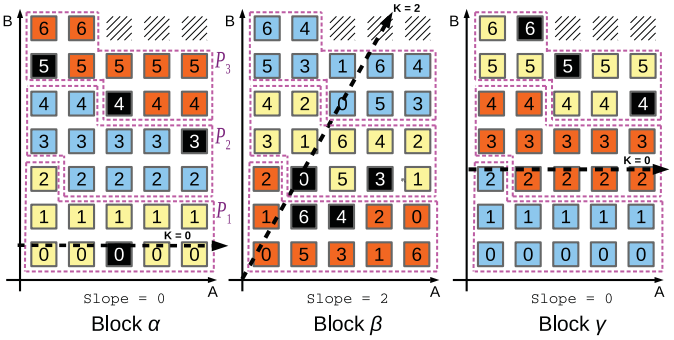
Similar to ECP, we can exploit block cooperation, allowing multiple blocks to work cooperatively to keep their associated page alive. In contrast to ECP, Aegis metadata is fixed and independent of the number of faults in a data block. Therefore, it is not possible to share metadata in order to exploit 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 Fig. 6b, three blocks are joined together and each portion of the plane (i.e., P_1, P_2 and P_3 in Fig. 6b) 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 a valid slope for the new date layout, more blocks gradually invited join the *Co-op* block to help. Block cooperation in Aegis helps to spread faults across different blocks more evenly, limited page waste due to early block failures. For example, in Fig. 6a, the number of faults are 6, 4, and 1 in the data blocks that change to 4, 4, and 3 in Fig. 6b, after applying data layout reorganization.

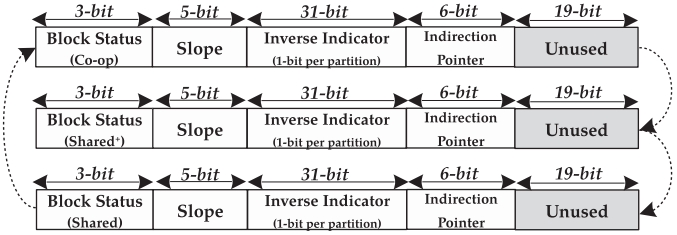
Assuming two blocks cooperates to save a faulty block, but no valid slope, K, is found for at least one of them, then the controller needs to select another private data block to perform



(a) Aegis with no block cooperation. No valid slope is found for block α which results in disabling entire the page.



(b) Aegis with multiple block cooperation. Blocks β and γ are joined to the dead block α to save the entire page. With data layout reorganization finding valid slope to revive the dead block is possible.

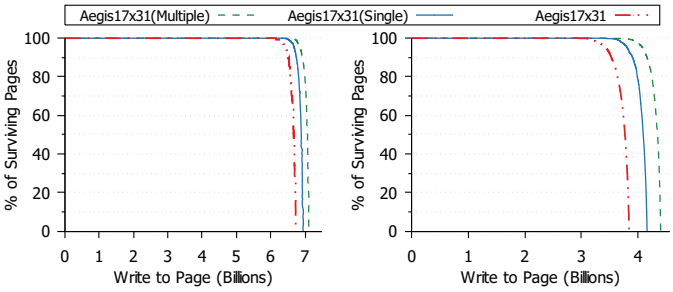


(c) the format of the metadata for each block in Aegis (17×31) with block cooperation.

Fig. 6. Integrating block cooperation in Aegis.

the join operation. The number of retries can be performed is equal to the number of private blocks within a page, or we can also select a predefined *maximum try threshold*. For instance, the block α in Fig. 6a 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 multiple 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, in order to manage the complexity of error correction. We define this variable as the *sharing level* in the data layout reorganization. Fig. 6c 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



(a) CoV=0.20 (b) CoV=0.30

Fig. 7. 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.

the inverse indicator for partitions. The inverse indicator bits denote whether the data is stored in its actual or inverted form. Fig. 7 shows page lifetimes with single and multiple levels of sharing integrated with Aegis, and compares against Aegis without the cooperative block. 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 set to 8 for multi-sharing. Moreover, we set a limit of 8 blocks that can cooperate in our reported results. The memory lifetime increases by more than 3% (6%), and 8%(14%) when single (multiple) sharing is used.

V. Discussion

Exploiting metadata sharing for ECP, and data layout reorganization for Aegis, improve memory lifetimes considerably. However, block cooperation adds some complexity to the memory system. 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 next memory access's latency to the subsequent block within the page (i.e., a row buffer hit is around 20ns [16]). 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 higher latency access (on the order of thousands or millions of cycles). Therefore, saving blocks from failure for the cost of some small latency is much more palatable, as compared to disabling the page.

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. 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.

VI. Conclusion

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, we proposed block cooperation technique, where other blocks help failed blocks to remain alive and extend the page's lifetime.

We combined the proposed technique with two error recovery schemes, ECP and Aegis, to increase memory lifetime. Employing block cooperation at a single level (or multiple levels) on top of Error Correction Pointer (ECP) and Aegis, increased memory lifetimes by 28% (37%), and 8% (14%) on average, respectively.

References

[1] ITRS, "More moore," 2015. [Online]. Available: http://www.itrs2.net/

[2] K. Kim *et al.*, "Technology for sub-50 nm dram and nand flash manufacturing," *IEDM Tech. Dig*, vol. 144, pp. 323–326, 2005.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 2–13.

[4] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo *et al.*, "A 20nm 1.8 v 8gb pram with 40mb/s program bandwidth," in *2012 IEEE International Solid-State Circuits Conference*. IEEE, 2012, pp. 46–48.

[5] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, "Multilevel-cell phase-change memory: A viable technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 87–100, 2016.

[6] W. Zhang and T. Li, "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, 2009, pp. 2–13.

[7] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 141–152.

[8] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "Safer: Stuck-at-fault error recovery for memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 115–124.

[9] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 433–444.

[10] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "Free-p: Protecting non-volatile memory against both hard and soft errors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 466–477.

[11] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie memory: Extending memory lifetime by reviving dead blocks," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 452–463.

[12] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *Proceedings of the 44th Annual International Symposium on Microarchitecture*. ACM, 2011, pp. 318–328.

[13] R. Melhem, R. Maddah, and S. Cho, "Rdis: A recursively defined invertible set scheme to tolerate multiple stuck-at faults in resistive memory," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE, 2012, pp. 1–12.

[14] S. Baek, H. G. Lee, C. Nicopoulos, and J. Kim, "A dual-phase compression mechanism for hybrid dram/pcm main memory architectures," in *Great lakes symposium on VLSI*. ACM, 2012, pp. 345–350.

[15] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proceedings of the 42nd Annual Symposium on Microarchitecture*. ACM, 2009, pp. 14–23.

[16] B. Jacob, S. Ng, and D. Wang, *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.