RETROFIT: Fault-Aware Wear Leveling

Jiangwei Zhang, Donald Kline Jr., Liang Fang, Rami Melhem, and Alex K. Jones

Abstract—Phase-change memory (PCM) and resistive memory (RRAM) are promising alternatives to traditional memory technologies. However, both PCM and RRAM suffer from limited write endurance and due to process variation from scaling, increasing number of early cell failures continue to put pressure on wear-leveling and fault tolerance techniques. In this paper, we propose RETROFIT, which leverages the spare "gap" row used as temporary storage in wear leveling to also be used strategically to guard against early cell wear out. RETROFIT is compatible with error correction schemes targeted at mitigating stuck-at faults and provides benefits when single or multiple spare rows are available. RETROFIT enhances lifetime by as much as 107 percent over traditional gap-based wear leveling and 8 percent over perfectly uniform wear leveling with a similar overhead. Furthermore, RETROFIT scales better than
wear-leveling combined with error correction as process variation increases.

Index Terms—Emerging memories, reliability, wear-leveling, and fault-tolerance

1 Introduction

Emerging memories such as Phase-change memory (PCM) and resistive memory (RRAM), which use their internal resistance to store data, are promising alternatives to scaled conventional memories as they provide improvements in density, energy, and non-volatility [1], [2]. Unfortunately, PCM and RRAM suffer from limited write endurance, becoming stuck-at a particular value, which for PCM occurs after 10^8 to 10^9 write cycles [2]. Wear-leveling techniques attempt to distribute writes that are inherently biased to relatively few rows, uniformly throughout the physical memory using logical to physical remapping [3].

Unfortunately, increased process variation, due to technology scaling [4], can lead to earlier cell wear-out from physical defects and transition pulse mismatches. Thus, even with perfectly uniform wear-leveling, "weak" cells will wear-out faster. For example, with an average lifetime of 10^8 writes per cell, for a coefficient of variation (CoV) of 0.25, the fault rate reaches 10^{-4} after 9 million writes, while for higher CoV of 0.30 and 0.35, this threshold is reached after only 2 and half million writes, respectively. These cell faults manifest as "stuck-at" faults, where a cell can be read but the state of the cell is immutable. Certain numbers of stuck-at faults can be handled by traditional error-correction codes (ECC) or tuned error correction schemes for stuck-at faults such as Error Correction Pointers (ECP) [5]. Alternatively, after the error correction capability is exceeded, row sparing can employ a spare row and retire the faulty row [3], [6].

- *J. Zhang is with the National University of Defense Technology, Changsha 410073, China, and is with the ECE Department, University of Pittsburgh, Pittsburgh, PA 15261. E-mail: jiz148@pitt.edu.*
- *L. Fang is with the National University of Defense Technology, Changsha 410073, China. E-mail: lfang@nudt.edu.cn.*
- *D. Kline Jr. and A. K. Jones are with the ECE Department, University of Pittsburgh, Pittsburgh, PA 15261. E-mail: {dek61, akjones}@pitt.edu.*
- *R. Melhem is with the CS Department, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: melhem@cs.pitt.edu.*

Manuscript received 24 Feb. 2018; revised 29 Apr. 2018; accepted 16 May 2018. Date of publication 23 May 2018; date of current version 25 June 2018.
(Corresponding author: Jiangwei Zhang.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2018.2840137

In this paper, we propose RETROFIT, a new technique to combine error correction, row sparing, and wear leveling to prolong memory lifetime. Reflecting this idea, RETROFIT, or *Realizing Extreme Endurance Through Row-sparing Optimized Wear-leveling For Improved Tolerance*, is more than a simple combination of these three ideas based on the following observations: First, ECP naturally records the number of faults in its row. This can be used as a measure of "premature aging" and indicate rows with weak cells due to variation. It can also be used to guide wear leveling to reduce writing activities to these rows. Second, traditional wear leveling, such as the start-gap (SG) method [3], naturally includes an additional row to allow logical data to move into different physical locations, which could also be employed in protecting rows with weak cells. Third, data movement in wear leveling is typically designed to occur at regular intervals to minimize write performance overheads. However, a biased movement pattern may provide more flexibility to protect prematurely aging rows before the row wears out. Finally, if a row does wear out, the additional row can serve as a spare row, at the expense of wear leveling, as a last ditch effort to keep the memory in service.

Based on these observations, RETROFIT uses ECP pointers for both error correction and to track memory aging. The row that exhibits the most faults is "quarantined" by the gap row included for wear leveling. This gap row cannot be accessed by programmatic memory writes for the majority of memory accesses, which protects the row from developing additional endurance faults. If multiple spare rows are available they are put into service to both help with wear leveling, and when needed to serve as quarantine rows. Only after a row exhibits more faults than can be corrected with ECP, is the row taken out of service entirely. These quarantines extend the memory lifetime by identifying rows containing weak cells prone to early failure but without knowing *a priori* the location and endurance of weak cells prior to the memory being placed in service.

Thus, RETROFIT makes the following contributions:

(1) A novel collaborative fault-tolerance design leveraging error-correction to optimize wear leveling to exceed the endurance of either solution alone.
(2) A method to take the benefit of sparing from inception rather than after failure to extend lifetime beyond wear leveling combined with row sparing.
(3) An architecture design and a detailed evaluation of this collaborative approach is compared to traditional gap-based wear leveling (SG), row sparing (RS), and perfectly uniform wear-leveling (UWL) using equivalent storage overhead.

RETROFIT improves lifetime by 107 percent for a gap-based wear leveling system without error correction, while also providing improvements with ECP. RETROFIT can actually exceed the lifetime of UWL. Moreover, its lifetime improvements hold up well while those of SG and RS degrade significantly for larger variation.

2 Related Work

Single error correction, double error detection (SECDED) [7] is the most popular ECC for transient fault protection in DRAM. ECP [5] is tuned specifically to tolerate stuck-at faults, using pointers to record the addresses of faulty bits along with a "spare" bit to store its value. SECDED is only effective for low stuck-at fault rates ($<10^{-6}$) and wears out encoding bits quickly. In contrast, ECP auxiliary bits are written infrequently when new faults are discovered, minimizing their wear out potential. As an alternative to ECP, Partition-and-flip (PAF) schemes attempt to partition faults within

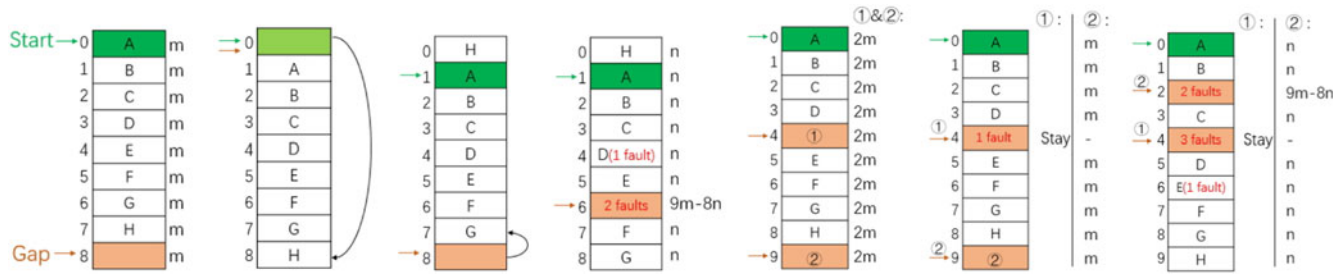

Fig. 1. Examples of RETROFIT on a memory containing 8 logical rows and one or multiple spare rows.

a row into different groups and optionally invert the groups to match the value to the fault [14]. Aegis [8] and Dynamic Aegis [15] use complex partitioning to better match values to stuck-at faults. However, ECP directly maintains the number of faults per row making it natural for RETROFIT.

SG uses a "gap" row and algebraic mapping from logical to physical rows for wear-leveling [3]. After a fixed number of writes occurs, one row is moved to its neighboring location by shifting the gap, ultimately cycling all logical rows through all physical rows. Bloom Filter wear-leveling (BWL) [9] records hot addresses swapping hot and cold rows to prolong memory lifetime. BWL also assumes expensive to discover *a priori* row-level endurance variation knowledge to map hot rows to "strong rows." Toss-up wear-leveling (TWL) [10] also relies on *page-level* endurance variation information to bond strong and weak pages.

Row sparing, where faulty rows are retired and replaced with spare rows, is also a vital technique for memory with low endurance [11]. Data dependent sparing (DDS) [6] postpones retirement by attempting to use the faulty row for writes in which the data matches the stuck-at value.

RETROFIT extends wear-leveling at the row granularity with collaboratively designed fault-tolerance to improve lifetime. This is accomplished without need for detailed foreknowledge of variation, which may be an impractical assumption at row-level granularity. RETROFIT, like other row-related wear leveling techniques, is also compatible with page-level wear-leveling techniques such as TWL.

3 RETROFIT

To explain RETROFIT we begin by reviewing gap-based wear leveling with an illustration. An example eight row memory page is shown in Fig. 1a. There are nine physical rows actually allocated for this page, including eight "logical" rows that store data, marked with 'A' through 'H,' and one "gap" row. A gap register is used to record the physical address of the gap and a start register points to the beginning of the logical rows to track the logical to physical mapping. The wear leveling is accomplished by moving the gap up to the neighboring physical address every m row writes—indicated in Fig. 1a to the right of each row—and copying the existing data to the previous gap location. Once the gap reaches the top physical row, it moves down to the bottom physical row, as shown in Fig. 1b, resulting in all of the logical rows and the start register being shifted down one physical row, as shown in Fig. 1c. We define this process—the gap leaving and returning to its original physical address—a *gap round*. Eventually the start register will return to its original address, in this example in eight gap rounds, completing a *wear leveling cycle*. Using this method simplifies logical to physical address translation to a series of addition operations based on the start offset and an increment if the resulting address is at or greater than the gap row location.

RETROFIT leverages the gap-wear leveling concept but utilizes the gap row to accomplish both wear leveling and row-sparing through non-uniform movement of the gap row within the gap round. In this way, RETROFIT utilizes the spare rows prior to row failure by protecting and extending the life of weak rows that wear out more quickly before failure. Prior to observing any faults, RETROFIT functions identically to regular gap wear leveling as previously described (i.e., Figs. 1a, 1b, 1c). The difference occurs after the first fault emerges. RETROFIT now observes this row with a fault to be prematurely aging and attempts to protect this row against receiving future writes. When a row emerges with more faults, RETROFIT starts guarding the newly discovered "weakest" row and puts the previously weakest row back into service. RETROFIT tracks this information by examining which row is protecting the most faults using their ECP auxiliary bits. RETROFIT can also function with ECC in place of ECP, however, it would require counters to track the number of faults per row. As shown in Fig. 1d, RETROFIT keeps the gap row in the location of the weakest row for nearly the entire gap round using nonuniform movement. In this example, the gap remains in position 6 for $9m$-$8n$ writes and only stays in position for every other location for n writes with the assumption that $n << m$. Thus, if m = 100 and n = 10 the gap remains at position 6 for 820 writes while remaining at every other row for only 10 writes. RETROFIT achieves this nonuniform movement while preserving both the gap round and the wear leveling cycle. Finally, if a row's faults exceed its fault protection, the gap replaces this row entirely at the expense of continuing wear leveling to further stave off potential failure.

To integrate spare rows into RETROFIT, it is our goal to put these rows into service immediately, rather than wait for rows to wear out. Thus, prior to the occurance of faults, we extend gap-based wear leveling to utilize multiple gap rows. To maintain the same wear leveling rate, gaps move every Sm writes (i.e., S times slower) where S is the number of gaps. However, after a gap round completes, the start register will have moved S rows, maintaining the same wear leveling rate. An example is shown in Fig. 1e with two gaps where both gaps move twice as slow (every $2m$ writes). After the first endurance fault emerges, multi-gap RETROFIT allocates a gap to completely "quarantine" that row and the remaining gaps continue wear leveling.

Multi-gap RETROFIT operates in two modes where $W < S$ and $W \geq S$, where W is the number of weak rows discovered through manifested faults. If $W < S$, RETROFIT quarantines these W rows with spares and increases the gap movement pace to $(S-W)m$ writes. If a gap row reaches a spare row it skips to the next address allowing the spare row to remain in place. An example of this case is shown in Fig. 1f where $S = 2$ and $W = 1$ where spare ① guards row 4 with one fault and spare ② continues wear leveling as a gap row with a pace of m. When $W \geq S$, the S-1 weakest rows are protected with spares and the Sth weakest row is guarded by RETROFIT with nonuniform motion as described in the single gap case. An example is shown in Fig. 1g, when $W = 3 > S = 2$. In this case, spare row ① quarantines the row with most faults (row 4), while spare row ② does uneven shifting to protect the row with the second most faults (row 2). This approach allows all spare rows to guard the maximal amount of weakest rows as soon as they are detected. Like in the single gap, when S rows have failed altogether, all spares replace failed rows at the expense of continuing wear leveling.

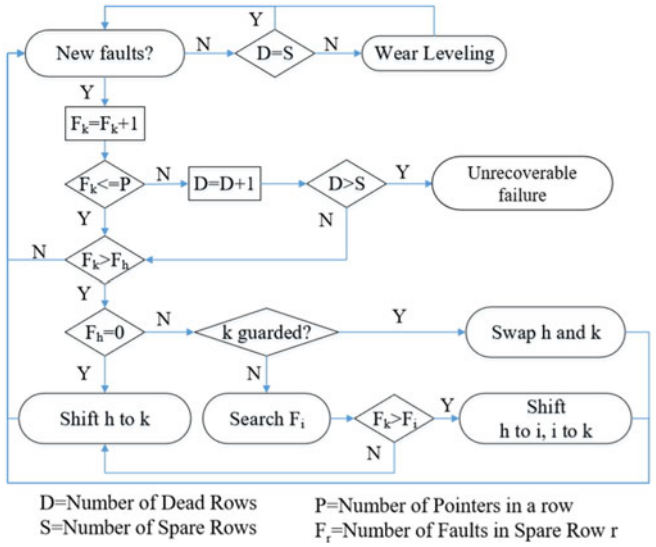

Fig. 2. RETROFIT spare/gap row allocation logic flow.

RETROFIT can be implemented through modification of the memory controller. Identical to traditional gap-based wear leveling, a record for each page with N memory rows must store $\log(N)$ bits for the start register. RETROFIT also must store $S\log(N+S)$ bits to store the locations of the S spare/gap rows, which is identical to traditional gap-based wear leveling if $S = 1$. RETROFIT expands the counter per page from $\log(m)$ to $\log((N+1)m)$, which for 4 KB pages with 512-bit cache lines (rows), adds six bits to the counter. The number of faults, F, present is equivalent to and can be obtained from the number of error correction pointers in service. For example, if the `full` bit is asserted, then F=P, otherwise, the first pointer encodes F. We assume F is known during a write operation, as F (and each fault' location) is required to encode the ECP bits. If F cannot be queried or ECC rather than ECP is used, it can be stored in cache with small overhead ($\leq$3 bits per cache line).

In Fig. 2, we show a logic flowchart for RETROFIT handling new faults. When a new fault appears during a write operation to row k, a new pointer is assigned to handle that fault and F_k is increased. If $F_k > P$, this physical row must be retired. To replace the retired (dead) row, a spare row (gap) that is not fixed to another retired row is shifted to this physical address. If $F_k \leq P$ the spare/gap row h that is covering the least faults F_h is determined. In the case of a tie row for F_h row h is the row with F_h guarded by the gap.

If $F_k \leq F_h$ the spare/gap row assignments are unchanged. If $F_k > F_h$ then one of three cases occurs. Case 1, $F_h = 0$, indicating that a spare/gap row is available, which is immediately assigned to k. Case 2, k is guarded by a gap and h is guarded by a spare. These roles are exchanged as h is the new minimum fault row needing to be guarded. Note, this infrequent case only occurs when a gap has temporarily left the row it is guarding to quickly complete a gap round through nonuniform movement. Case 3, k is unguarded and h is guarded by the gap. In this case, spare row i guarding the next least faults F_i is examined. If $F_k > F_i$ then k assumes the spare and i takes the gap from h. Otherwise k takes the gap from h as i has been at this fault level longer than k, so i is likely weaker than k. Either way, h becomes unprotected. If any new fault appears when transferring a row back into service, the system repeats this process. Ultimately with no new faults appearing, the write completes and new writes to memory and wear leveling resume.

4 Evaluation

To validate RETROFIT, we compare with RS [11], SG [3], and BWL [9]. For comparison with BWL, we report BWL's most optimistic case which is perfectly uniform wear leveling or UWL, i.e., all writes are equally distributed in the page. For both SG and UWL we make comparisons with equal storage overheads by including traditional spare rows that guard rows with more faults than can be corrected, labeled as SG+RS and UWL+RS, respectively. In our experiments, different levels of ECP protection are provided. We assume the lifetime of each cell within the memory follows a normal distribution with a mean value of 10^8 writes and a coefficient of variance of 0.30. Our system uses 4 KB pages, 512-bit cache lines (64 lines per page), and employs a 4 MB 16-way set associative last lavel cache. We set m = 100 and n = 5 and verified this configuration had similar performance to SG in simulation. We use a 13-bit counter per page to record the number of row writes. We also enforced a perfectly uniform intra-line wear-leveling for all inter-line wear leveling approaches.

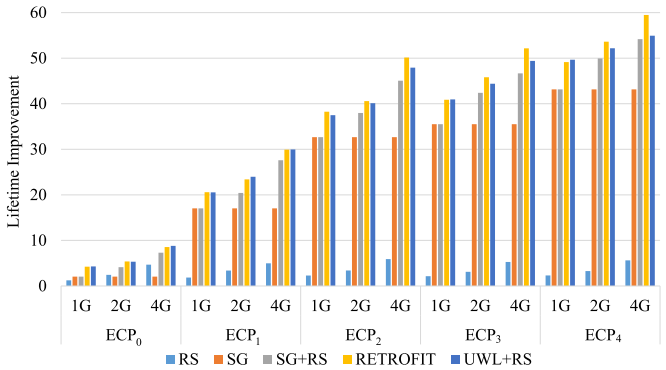

Fig. 3. Lifetime of the memory protected with CoV=0.30.

To stimulate our memory we use a combination of address traces from eight representative SPEC CPU 2006 [12] benchmarks: gcc, mcf, namd, fft, bzip2, libquantum, milc, and sjeng. These traces were recorded using a Pintool [13]. We report lifetime based on when the first unrecoverable failure is reported within any page. Each scheme was tested against ten different endurance maps with the same CoV and the average lifetime across all ten tests is reported.

Fig. 3 shows the memory lifetime improvements of RS, SG, SG+RS, RETROFIT, and UWL+RS normalized to a baseline memory without wear leveling but with equivalent ECP protection. The results are shown for one (1G), two (2G), and four (4G) gaps with zero to four ECP pointers. RS performs poorly alone, while SG is very effective over the baseline except compared to RS with two and four spares for ECP_0. In all cases SG+RS provides no appreciable benefit for a single gap system but provides some improvements with two and four spares. RETROFIT provides better lifetimes than RS, SG, and SG+RS with a similar space overhead in all cases. This translates to a 14-107 percent improvement over SG with one gap, a 7-30 percent improvement over SG+RS. RETROFIT performs similarly, within -3 to 2 percent of the lifetime, to UWL+RS for $P < 2$. When $P \geq 2$ and $S > 1$, RETROFIT always yields a superior lifetime to UWL+RS. For example, when there are four spare rows and ECP_4 is applied, RETROFIT has an 8 percent longer lifetime than UWL+RS due to its early discovery and protection of weaker rows.

To study the impact of CoV on these wear leveling approaches, we performed experiments with lower CoV = 0.25 and higher CoV = 0.35 reported in Figs. 4 and 5, respectively. The comparisons follow a similar pattern as with CoV = 0.30, however, the improvements of RETROFIT are amplified as the CoV increases. For example, as CoV increases from 0.25 to 0.35, at $P = 4$ and $S = 4$, SG, SG+RS, and UWL+RS experience a 40, 16, and 13 percent drop, respectively. In contrast, RETROFIT's early protection of weak rows attenuates its lifetime degradation to only 10 percent. Thus, as variation increases the improvement of RETROFIT over even perfectly uniform wear leveling (variation oblivious) will increase.

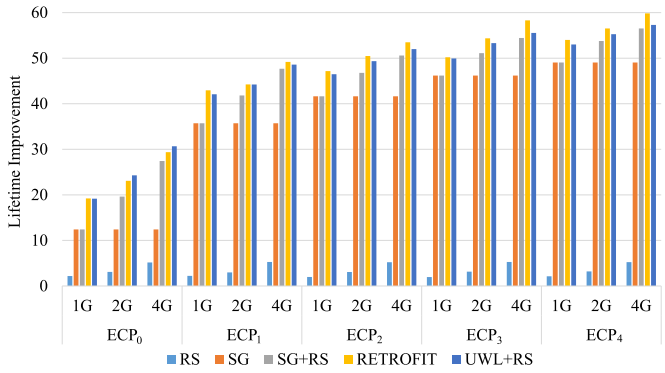

Fig. 4. Lifetime of the memory protected with CoV=0.25.

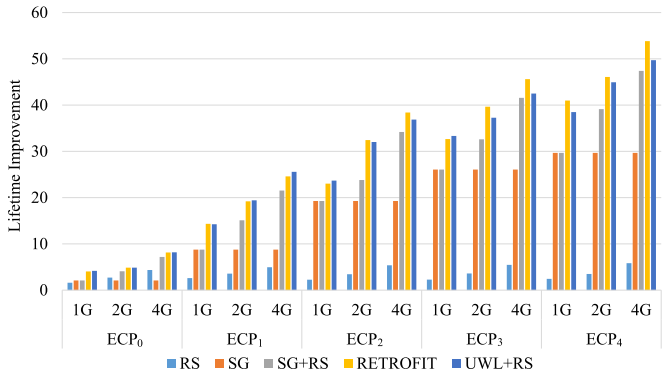

Fig. 5. Lifetime of the memory protected with CoV=0.35.

5 Conclusion

The collaborative design of RETROFIT combined wear-leveling, row sparing, and error correction techniques into a single approach that, with minimal enhancement, can leverage systems that utilize traditional gap-based wear leveling and memory with ECP. RETROFIT extends memory lifetimes *without a priori* knowledge of the location and endurance of weak cells. It also provides dramatic lifetime improvements even when spare rows are not available and scales well as the CoV increases as expected with deeply scaled semiconductor fabrication. While, RETROFIT does not degrade performance over SG we plan to conduct a detailed performance evaluation and consider pareto-optimal points for design tradeoffs in future work.

References

[1] O. Zilberberg, S. Weiss, and S. Toledo, "Phase-change memory: An architectural perspective," *ACM Comput. Surv.*, vol. 45, no. 3, 2013, Art. no. 29.

[2] C. J. Xue, G. Sun, Y. Zhang, J. J. Yang, Y. Chen, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *Proc. 9th Int. Conf. Hardware/Software Codes. Syst. Synthesis*, 2011, pp. 325–334.

[3] L. Jiang, Y. Du, Y. Zhang, B. R. Childers, and J. Yang, "LLS: Cooperative integration of wear-leveling and salvaging for PCM main memory," in *Proc. IEEE/IFIP 41st Int. Conf. Depend. Syst. Netw.*, 2011, pp. 221–232.

[4] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," in *Proc. IEEE Int. Electron Devices Meeting*, 2005, pp. 323–326.

[5] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 141–152.

[6] R. Maddah, S. Cho, and R. Melhem, "Data dependent sparing to manage better-than-bad blocks," *IEEE Comput. Archit. Lett.*, vol. 12, no. 2, pp. 43–46, Jul.–Dec. 2013.

[7] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950.

[8] 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 *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2013, pp. 433–444.

[9] J. Yun, S. Lee, and S. Yoo, "Bloom filter-based dynamic wear leveling for phase-change RAM," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2012, pp. 1513–1518.

[10] X. Zhang and G. Sun, "Toss-up wear leveling: Protecting phase-change memories from inconsistent write patterns," in *Proc. 54th ACM/EDAC/IEEE Des. Autom. Conf.*, 2017, pp. 1–6.

[11] S.-K. Jo, "Flash memory device for performing bad block management and method of performing bad block management of flash memory device," U.S. Patent 7,434,122, Oct. 7, 2008.

[12] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, 2006.

[13] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 190–200, 2005.

[14] J. Zhang, D. Kline Jr., L. Fang, R. Melhem, and A. K. Jones, "Yoda: Judge me by my size, do you?," *IEEE Int. Conf. Comput. Des. (ICCD)*, 2017, pp. 395–398.

[15] J. Zhang, D. Kline Jr., L. Fang, R. Melhem, and A. K. Jones, "Dynamic partitioning to mitigate stuck-at faults in emerging memories," in *Proc. 36th Int. Conf. Comput.-Aided Des.*, 2017, pp. 651–658.