Wear Leveling for Crossbar Resistive Memory

Wen Wen
Department of Electrical and Computer Engineering
University of Pittsburgh
wew55@pitt.edu

Youtao Zhang
Department of Computer Science
University of Pittsburgh
zhangyt@cs.pitt.edu

Jun Yang
Department of Electrical and Computer Engineering
University of Pittsburgh
juy9@pitt.edu

ABSTRACT

Resistive Memory (ReRAM) is an emerging non-volatile memory technology that has many advantages over conventional DRAM. ReRAM crossbar has the smallest $4F^2$ planar cell size and thus is widely adopted for constructing dense memory with large capacity. However, ReRAM crossbar suffers from large sneaky currents and IR drop. To ensure write reliability, ReRAM write drivers choose larger than ideal write voltages, which over-SET/over-RESET many cells at runtime and lead to severely degraded chip lifetime.

In this paper, we propose XWL, a novel table based wear leveling scheme for ReRAM crossbars. We study the correlation between write endurance and voltage stress in ReRAM crossbar. By estimating and tracking the effective write stress to different rows at runtime, XWL chooses the ones that are stressed the most to mitigate. Our experimental results show that, on average, XWL improves the ReRAM crossbar lifetime by 324% over the baseline, with only 6.1% performance overhead.

CCS CONCEPTS

• **Computer systems organization → Processors and memory architectures**; *Embedded systems*; • **Hardware → Memory and dense storage**;

KEYWORDS

Resistive memory, crossbar array, wear leveling, endurance

ACM Reference Format:
Wen Wen, Youtao Zhang, and Jun Yang. 2018. Wear Leveling for Crossbar Resistive Memory. In *DAC '18: DAC '18: The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196138

1 INTRODUCTION

Modern applications, e.g., big data analytics, video streaming and graphical games, exhibit increasing demand for large capacity memory. However, DRAM, the *de facto* choice for main memory, faces low density, short refreshing interval and scalability challenges at 20nm and beyond [10]. ReRAM (Resistive Memory) has recently emerged as a promising candidate for constructing future large capacity main memory [16, 18, 19, 21]. It has many advantages such as non-volatility, no refreshing, high density and almost-zero standby power. Comparing to other non-volatile memory technologies, ReRAM has better density and scalability than those of STT-MRAM, and better write performance than that of PCM. However, ReRAM suffers from unsatisfactory write endurance [23]. Recent studies showed that the endurances of ReRAM chips adopting different resistive materials range from 10^3 to 10^9 [20].

ReRAM cell arrays often adopt the crossbar architecture to achieve the smallest $4F^2$ planar cell size [21]. ReRAM crossbars enable the construction of dense main memory with large capacity, but face large sneaky currents and IR drop issues [16, 18, 22] — the leakage currents flowing through half-selected cells during writes are not negligible. Adopting access diodes helps to mitigate the issue, but cannot eliminate it completely.

To mitigate sneaky current and IR drop in ReRAM crossbars, ReRAM writes, in particular, RESET operations, conservatively adopt the worst-case latency. Recent studies have optimized the write latency from one latency fitting all cells in the crossbar to different latencies based on row address, i.e., writing the rows that are close to the write drivers can finish faster due to smaller IR drop on their cells [21, 22]. Considering the data patterns inside the cell array can further improve the average write latency [18], resulting in significantly improved write performance for ReRAM crossbars.

However, prior studies [3, 8, 13] showed that programming ReRAM cells with longer than necessary pulse length over-SETs or over-RESETs the corresponding cells, leading to orders of magnitude degradation in ReRAM cell lifetime [3]. While optimized write strategies [18, 21, 22] write different rows using different write latencies, the rows being close to the write drivers still get stressed more than others. Adopting traditional wear leveling techniques that evenly distribute writes across all rows in ReRAM space would become less effective — the rows that close to the drivers are approaching their lifetime while others may still have a lot of endurance to use. Thus, it is important to devise a wear leveling approach that considers the stress difference at runtime.

In this paper, we propose XWL, a novel table based wear leveling design for addressing the write endurance degradation from IR drop in ReRAM crossbars. We summarize our contributions as follows.

- We study write endurance variation in ReRAM crossbar, which reveals that the effective write, i.e., the actual degree of ReRAM wearing out, depends on data patterns and row addresses at runtime. To the best of our knowledge, this is the first study revealing the unique wearing characteristic in ReRAM crossbars.
- We propose XWL, a novel table based wear leveling design that tracks the effective writes at runtime. XWL periodically remaps the ReRAM rows that are stressed the most, rather than the ones accumulating the most write counts.

DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196138

- We evaluate the proposed wear leveling scheme. The experimental results reveal that, our design improves write endurance by 324%, compared to the baseline design.

In the rest of the paper, we introduce backgrounds and motivations in Section 2. We elaborate the wear leveling scheme in Section 3. We present the evaluations in Section 4. We discuss prior related work in Section 5 and conclude the paper in Section 6.

2 BACKGROUNDS AND MOTIVATIONS

2.1 ReRAM Basics and Crossbar Structure

As Figure 1 (a) shows, a ReRAM cell has two metal layers, and an oxide layer that is sandwiched between the metal layers. A ReRAM cell uses two resistance states, i.e., low resistance state (LRS) and high resistance state (HRS), to represent logical '1' and '0', respectively. Programming a ReRAM cell is to apply a write voltage with appropriate pulse width and magnitude to the cell, which switches the cell's resistance state either from HRS to LRS, or from LRS to HRS, referred to as SET and RESET, respectively.

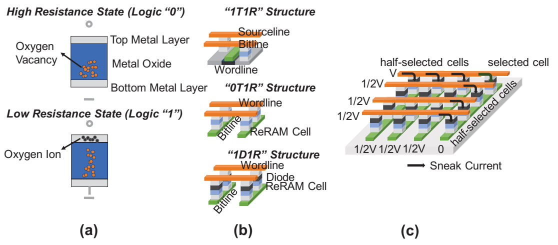


Figure 1: The ReRAM basics: (a) the cell structure; (b) the three typical ReRAM array structures; and (c) the sneak current issue in ReRAM crossbar array.

A wide range of metal oxide materials, such as HfOx-based and TiOx-based materials, have been proposed to construct ReRAM cells. According to previous studies, the ReRAM cells using different materials present different energy, scalability, and most importantly, write endurance characteristics. The techniques developed in this paper are generally applicable to all kinds of ReRAM cells.

There are three typical ReRAM array structures, as shown in Figure 1 (b). The 1T1R structure has an access transistor, which is similar to that of conventional DRAM cells. It has largest cell size. Both of 0T1R and 1D1R structures are fabricated as ReRAM crossbar arrays that have the smallest $4F^2$ planar cell size. The difference is that the 1D1R structure adopts an access diode that helps to reduce sneak currents in the crossbar. In this work, our ReRAM crossbar adopts 1D1R cell structure.

2.2 Motivation

We next study the IR drop issue in the crossbar, and analyze its impact on ReRAM cell write endurance.

2.2.1 *IR Drop Issue.*

Writing a ReRAM line with multiple cells consists of two steps: a SET phase to write 1s, and a RESET phase to write all 0s. As shown in Figure 1 (c), to program one cell in ReRAM crossbar, e.g., a SET or RESET operation, a write driver activates several cells along a wordline by applying with V_{WRITE} voltage, while the voltage bias of bitlines that have selected cells is set to 0V. In order to fully switch resistance state, these *selected cells* have the largest voltage stress. In contrast, for all other bitlines and wordlines, the voltage bias is set to V_{WRITE}. These ReRAM cells can be further categorized into *half-selected cells* that are on the selected bitlines and wordline, and *unselected cells* that are the rest of cells in the ReRAM crossbar. Ideally, there is no voltage stress on unselected cells. Prior work shows that SET operation are much faster when compared to RESET operation, therefore the long RESET latency dominates the write timing [18, 21, 22].

Previous reports showed that there are large sneaky currents flowing through half-selected cells in ReRAM crossbars, even after adopting diode selectors, while the sneak currents on unselected cells are negligible. These sneaky currents lead to large leakage power, and introduce significant voltage drop, i.e., IR drop, along the long wordlines and bitlines. With fast technology scaling, future ReRAM crossbars would have larger array size and larger wire resistance such that IR drop issue tends to worsen. The IR drop issue exists in all crossbar based memory architectures.

2.2.2 *Endurance variation in ReRAM crossbar.*

A recent study [23] revealed a tradeoff between write latency and endurance of ReRAM cell — the endurance degrades when write latency increases. The relationship can be analytically modeled using the following equation:

$$Endurance \approx (\frac{t_W}{t_0})^C \tag{1}$$

where t_W is write latency, t_0 and C are constants. In this paper, we choose the same $C = 2$ as in [23] to model a quadratic correlation between write endurance and latency.

Recent studies [18, 22] has shown that IR drop results in RESET latency discrepancy among the ReRAM cells due to different physical locations and dynamic bitline data patterns. According to Equation 1, the cells in ReRAM crossbar would exhibit endurance discrepancy.

To study the endurance discrepancy in a ReRAM crossbar, we use HSPICE to build a 512 × 512 crossbar model. We adopt the publicly available Verilog-A ReRAM cell model from [9], and integrate *DSGB* [21] to mitigate voltage discrepancy. The parameters of our model is presented in Table 1. Figure 2 summarizes the endurance discrepancy across the crossbar. We divide 512 rows to eight address groups with each group containing consecutive 64 rows. *Row Address Group 0* is the one that is the closest group to the write drivers. *LRS cell ratio* indicates the percentage of LRS cells in one bitline. We adopt the worst-case voltage drop and RESET latency in every 64 rows to represent one *Row Address Group*.

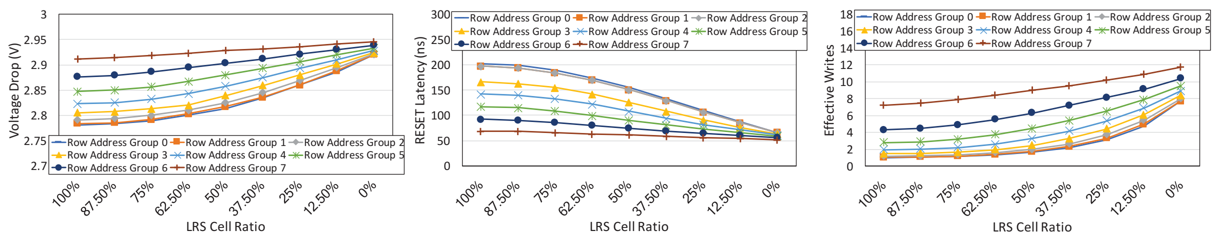
Table 1: Parameters in ReRAM Crossbar Modeling

Metric	Description	Value
A	Mat Size: A wordlines × A bitlines	512 × 512
n	Number of bits to read/write	8
I_{on}	RESET current of a LRS Cell	$88\mu A$
R_{WIRE}	Wire resistance between adjacent cells	2.82Ω
K_r	Nonlinearity of the selector	200
V_{WRITE}	Full selected voltage during write	3.0V
V_{READ}	Read voltage	1.5V

From Figure 2 (a) and (b), the more LRS cells on selected bitlines, the larger sneak current flows through half-selected cells. Thus we observe smaller voltage drop and longer RESET latency. Also, the farthest rows from write drivers are more vulnerable to the impact of bitline data patterns on RESET latency. The observation is similar to that in [18, 22]. In conclusion, the discrepancy of RESET latency leads to write endurance variation in ReRAM crossbar.

2.2.3 *Effective Write.*

In this paper, we use **effective write** to summarize the overall wearing effect of one write at runtime. Intuitively, let us assume that one cell can sustain 10^5 times writes if



(a) (b) (c)

Figure 2: Subfigures show that the variations of (a) voltage drop on selected cells and (b) RESET latency and (c) effective writes at different LRS cell percentages in bitlines when accessing to different row address in ReRAM array. The Row Address Group 0 represents farthest rows from drivers, and Row Address Group 7 consists of nearest rows to the drivers.

using write pulse width X and 10^6 times writes if using write pulse width Y. Assume other conditions are the same. We conclude that each write with pulse X corresponds to ten writes with pulse Y. According to Equation 1, the effective write depends on the write pulse width while an optimized write strategy [18] chooses pulse width based on (1) target row address and (2) the numbers of LRS cells in the bitline. Therefore, the actual effective write depends on the latter two factors.

Figure 2 (c) depicts the relationship between effective writes and row addresses and LRS ratios. In our experiments, when writing *Row Address Group 0* with 100% LRS cell ratio, the write takes longest duration to complete. Such a write has the smallest wearing effect, as shown in Equation 1. We normalize all other writes to this baseline, that is, the effective write of writing address group 0 under 100% LRS cell ratio is the normalized '1'. For all other writes, we calculate the effective writes with following equation:

$$EW = \left\lceil \left(\frac{t_L}{t}\right)^2 \right\rceil \tag{2}$$

where t_L is the longest write latency (i.e., writing group 0 with 100% LRS ratio); and t is the actual write latency of the given write.

2.2.4 Design Challenge. Given that writes to ReRAM crossbar exhibit different effective writes at runtime, to extend chip lifetime, we should evenly distribute effective writes across all ReRAM cells. Unfortunately, existing wear leveling approaches evenly distribute raw writes across all ReRAM cells. As a result, it is highly possible that rows in the address group 7 are worn out while the rows in the address group 0 are very healthy.

There are two families of wear leveling schemes: one is to track writes to blocks using a table and periodically mitigate the block that is stressed the most [6, 24, 25]; the other is having physical addresses randomly mapped to device addresses and periodically changes to a new random mapping [14, 15]. In this paper, we propose a table based wear leveling scheme that evenly distributes effective writes at runtime. We leave the development of randomized mapping based wear leveling on effective writes as our future work.

3 XWL: WEAR LEVELING FOR CROSSBAR RERAM MEMORY

3.1 An Overview

The workflow of XWL follows typical table-based wear leveling schemes, which consists of three stages: prediction, address remapping & data swapping and running, as shown in Figure 3. These three stages repeat in every interval, i.e., a number of writes.

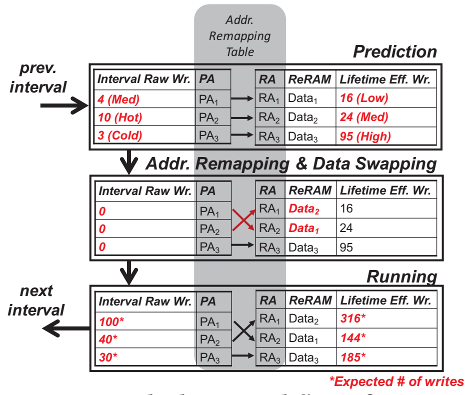

Addr. Remapping Table
Prediction
prev. interval
Interval Raw Wr. | PA | RA | ReRAM | Lifetime Eff. Wr.
4 (Med) | PA1 | RA1 | Data1 | 16 (Low)
10 (Hot) | PA2 | RA2 | Data2 | 24 (Med)
3 (Cold) | PA3 | RA3 | Data3 | 95 (High)
Addr. Remapping & Data Swapping
0 | PA1 | RA1 | Data2 | 16
0 | PA2 | RA2 | Data1 | 24
0 | PA3 | RA3 | Data3 | 95
Running
next interval
100* | PA1 | RA1 | Data2 | 316*
40* | PA2 | RA2 | Data1 | 144*
30* | PA3 | RA3 | Data3 | 185*
*Expected # of writes

Figure 3: The basic workflow of XWL.

XWL splits the whole ReRAM space into chunks and tracks writes to each chunk. In this paper, one chunk is a page. We differentiate two addresses in the following discussion. **Physical address (PA)** refers to the address after OS page table mapping. **Raw address (RA)** refers to the device address where the data are actually saved. As shown in Figure 3, XML attaches one *interval* entry to each PA chunk and one *lifetime* entry to each RA chunk.

In prediction stage, XWL tracks the number of writes to each PA chunk in the corresponding *interval* entry and the number of lifetime effective writes to each RA chunk in its *lifetime* entry. The major difference between XWL and conventional wear leveling is, instead of tracking raw write accesses for both tables, XWL records effective writes to update the $lifetime$ table and *raw writes* to update interval write table.

In address remapping & data swapping stage, XWL chooses one RA chunk and one PA chunk that are not mapped to each other. The choice involves two pairs, we change their PA to RA mapping accordingly. For example, in Figure 3, we choose PA-chunk-2 and RA-chunk-1, since PA-chunk-1 maps to RA-chunk-1, and PA-chunk-2 maps to RA-chunk-2, the swap results in PA-chunk-1 maps to RA-chunk-2 and PA-chunk-2 maps to RA-chunk-1, as shown in the figure. The candidate selection policy determines what pages are chosen to get remapped. We will present different algorithms in the next section. The design is to map hot physical pages to the ReRAM pages with the least degree of wearing out, similar to those previously design table-driven wear leveling algorithms [24]. Remapping involving reading two blocks and write two blocks. Clearly, the bigger the chunk is, the larger overhead the swap is. XWL cleared the interval entries after the swap.

In the running stage, each ReRAM page tracks incoming writes with predicted distribution, matching hot pages to low wearing out domains and cold pages to high wearing out domains, which achieves the aim of enhancing lifetime for overall crossbar ReRAM memory. During the running phase, both of two write tables keep updating with new write operations.

3.2 Design Details

3.2.1 Effective and Raw Write. In the proposed XWL scheme, we track both of the number of effective writes and raw writes at runtime. The effective write total of each chunk indicates how much lifetime the corresponding chunk has experienced while the raw write reflects the intrinsic access patterns of applications. The raw count would not change if having PA chunk remapped to a different RA location. However, their effective write counts depend on mapping. We use the number of raw writes in each interval to indicate how many incoming writes will reach to each ReRAM page. In contrast, to determine the degree of wearing out of each page, we need to adopt the proposed effective write for lifetime write table since it measures how many more writes each page can undertake before failures.

3.2.2 Updating Write Tables. While it is straightforward to update the interval raw write table, i.e., increment after each read or write, to update the lifetime table, we adopt Equation 2 and compute the effective write based on the write pulse width. Figure 4 illustrates the profiling scheme we used for dynamic RESET latency as well as updating effective write table.

As we adopted the bitline data pattern profiling and dynamic RESET latency in [18], the RESET latency is determined by row addresses and runtime bitline data patterns. In order to ensure the correctness of write timing, we conservatively assume each write after profiling always introduce one more LRS cell on the worst-case bitline, which prolongs the RESET latency. Similarly, we also have conservative assumption of updating effective writes. However, in contrast to dynamic RESET latency, we assume writes will bring more HRS cells instead, since more HRS cells lead to larger voltage drop on selected cells. Therefore, we have to track the worst-case LRS cell ratio to look up dynamic RESET timing as well as worst-case HRS cell ratio to update effective write table. In the case shown in Figure 4, in one row address group n of a simplified ReRAM crossbar, the worst-case LRS cell number is 5, and the worst-case HRS cell number is 3, both of which are incremented by 1 for each write request after profiling.

The dynamic RESET timing is simply determined by the table shown in Figure 2(b), which maps LRS cell ratio in a particular row address group to a conservative RESET timing. For discussion purpose, we assume the RESET latency is t_R in this case. As we want to RESET multiple cells, e.g. at most 8 bits in our design, within one ReRAM crossbar, the t_R is most conservative RESET timing to ensure write success, but it is too aggressive to use this latency to estimate effective writes with Equation 2. This is since the t_R may be too long for other bits that have larger voltage drop on selected cells owing to more HRS cells on their bitlines. When we RESET all bits with same t_R, those victim cells that take much longer RESET time than ideal one may be over-RESET, which leads to a endurance degradation. Therefore, we need to calculate the most conservative effective writes by using following formula:

$$EW_{addr} = EW_0 \cdot e^{C \cdot (V - V_0)} \tag{3}$$

where EW_{addr} is the most conservative effective writes at address $addr$, EW_0 is $\left\lceil (\frac{t_R}{t})^2 \right\rceil$ with RESET timing t_R, V_0 is the voltage drop at the worst-case LRS cell ratio, and V is the one at worst-case HRS cell ratio, and C is a fitting constant. Equation 3 is derived from experimental data of the different over-RESET voltages with same RESET pulse width on endurance degradation in [3].

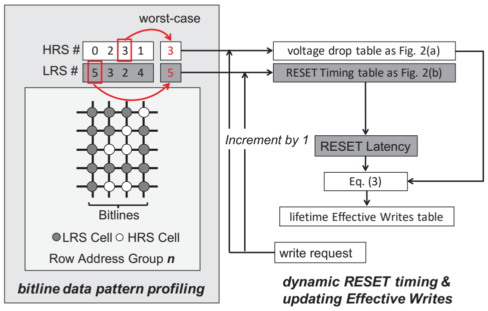

worst-case
HRS #
0 2 3 1
3
LRS #
5 3 2 4
5
Bitlines
● LRS Cell ○ HRS Cell
Row Address Group n
bitline data pattern profiling
voltage drop table as Fig. 2(a)
RESET Timing table as Fig. 2(b)
Increment by 1
RESET Latency
Eq. (3)
lifetime Effective Writes table
write request
dynamic RESET timing & updating Effective Writes

Figure 4: Profiling bitline data pattern for (1) optimized RESET latency and (2) estimating effective writes.

3.2.3 Address-Remapping Algorithm. As write tables are updated for each interval in memory controller, the physical addresses from CPU need to remap to ReRAM page real addresses while migrating data accordingly. As evaluated in experiment section, the naïve wear leveling technique, which simply remaps PA with largest raw writes to RA with smallest number of effective writes, helps to improve lifetime of ReRAM crossbars to certain extent. However, obviously this scheme ignores the fact that all pages are not worn out equally, and they actually depend on dynamic bitline data patterns and physical locations. Therefore, with only taking write access patterns of applications into consideration, it may be not able to effectively leverage incoming writes after address remapping.

In addition to raw write access patterns, we also want to exploit the impact of ReRAM crossbar features on endurance for address remapping. We introduce the *weights* to indicate the tendency of remapping a PA to a physical ReRAM crossbar page. Figure 5 illustrates our address remapping scheme. In this example, we partition ReRAM crossbar into 5 address groups. According to preceding discussion, the closer the group is from the write drivers, the more stress its cells accumulate from each write. Therefore, each group is assigned a different weight as follows.

$$weight_{addr} = \frac{\sum_{r=0}^{n-1} EW^r_{addr}}{n} \tag{4}$$

where $weight_{addr}$ is the weight at page address $addr$ and EW^r_{addr} is the effective writes at page address $addr$ with LRS cell ratio of r. It is worth noting that we average effective writes at same address with n different LRS cell ratios. This is since the data pattern can significantly change after prediction with much longer interval (10^4 writes) than profiling (64 writes), and we are no longer able to exploit bitline data pattern to estimate actual wearing out for future writes.

Moreover, we adopt the *Predict Write*, which estimates upper limit of effective writes if all writes reach to a particular page. It can be calculated by following equation:

$$PredictWr_{addr} = EW_{addr} + weight_{addr} \times interval \tag{5}$$

where $PredictWr_{addr}$ is the *Predict Writes* at page address *addr* and $interval$ is a parameter of how many writes between an address remapping.

Finally, as Figure 5 shown, the PA with the largest number of raw writes remaps to RA with smallest *predict writes* instead of effective writes, and vice versa.

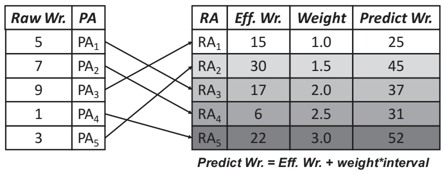


Figure 5: An example of PA to RA address remapping.

3.2.4 Design Overhead. XWL adds two tables with two entries per 4KB data chunk — we use 20 bits and 14 bits for the *effective writes* and *interval raw writes* counter, respectively. We add one 16-bit remapping entry for each chunk. The total space overhead is approximately $50bits/4KB = 1.56 \times 10^{-3}$. We assume the optimized write scheme exploits the LRS cell ratio information [18]. If not, adding online profiling introduces negligible overhead, as shown in [18]. We use CACTI [26] to model the two tables as direct mapped cache, the area and energy overheads are also negligible.

3.3 Process Variation Issue

We do not consider process variation (PV) in this paper. When taking PV into consideration, some of cells/rows would be more vulnerable to write operations than others. Several PV aware wear leveling techniques [6, 24, 25] have been recently proposed to mitigate this issue. XWL is a table based wear leveling scheme, which has the ability to address PV more flexibly. These designs are orthogonal to XWL in the paper.

4 EVALUATIONS

In this section, we first present our experimental setup, and then demonstrate the effectiveness of the proposed XWL scheme in endurance improvement. Finally, we estimate the data swapping overhead of XWL in performance.

4.1 Experiment Setup

In section 2.2, we model and simulate a 512×512 ReRAM crossbar to investigate the correlation between RESET latency and *effective writes*. In addition, we used an in-house architectural Chip Multiprocessor simulator to evaluate the proposed XWL scheme and compare it with baseline as well as naïve design. The system configuration is presented in Table 2. We used Pintool [12] to collect memory access traces from PARSEC [2], BioBench [1] and SPEC2006 [7] benchmark suites. All benchmarks are executed with or without wear leveling until first ReRAM page is worn out. We also use *flip-n-write* [5] to reduce the number of written bits. With a representative ReRAM device, we assume the ReRAM cell endurance is 1.6×10^6. For the proposed XWL, the default interval is 10^4 while we also evaluate different intervals in experiments. The benchmarks are characterized in Table 3 with write bandwidth to ReRAM memory. We adopt the profiling approach and dynamic RESET latency from [18].

In the paper, we compared the following wear leveling schemes:

- `NoWL`: baseline scheme, which adopts dynamic RESET latency and data pattern profiling, does not use any wear leveling techniques.
- `Naïve`: the wear leveling scheme, which follows the workflow introduced in Section 3.2, does not use proposed address remapping algorithm.
- `XWL`: the proposed wear leveling design.

Table 2: System Configuration

Processor	4 cores@1.8Ghz; single issue in-order CMP
L1 I/D-cache	Private; 16KB/core; 4-way; 2 cycles
L2 cache	Private; 1MB/core; 8-way; 64B; 10 cycles

Main memory	2Gb ReRAM; 4KB page; 64B per line;
	1 rank; 8 chips/rank;8 banks/chip;
	128 mats/bank;

ReRAM Timing	Read Latency 18ns@1.5V; SET latency 10ns@3V; RESET latency based on profiling@-3V

Table 3: Benchmark Summary

Name	Benchmark Suite	Write Bandwidth to ReRAM (MBps)
ferret	PARSEC	139.0
fasta_dna	BioBench	129.4
GemsFDTD	SPEC2006	123.2
bzip2	SPEC2006	61.3
zeusmp	SPEC2006	60.8
gcc	SPEC2006	56.6

4.2 Results

4.2.1 Endurance Improvement. Figure 6 presents the endurance improvements (normalized to `NoWL`). On average, by applying the proposed wear leveling techniques, we observed the significant endurance improvements by 285% and 324% for `Naïve` and `XWL`, respectively. Moreover, the proposed wear leveling `XWL` shows 14% more lifetime enhancement. In conclusion, by using proposed concept of *effective write* as well as the address remapping algorithm, the lifetime of crossbar ReRAM memory is effectively improved.

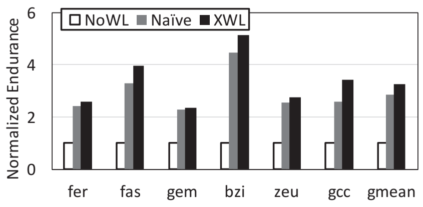


Figure 6: Comparison of normalized endurance.

To evaluate the impact of interval length, Figure 7 compares the normalized endurance improvements with different intervals, i.e., 10^4, 5×10^4 and 10^5. From the figure, the effectiveness of endurance improvement diminishes as interval gets longer for most benchmarks. On average, the normalized endurance improvements by using `XWL` with intervals of 10^4, 5×10^4 and 10^5 are 324%, 216% and 166%, respectively. This indicates that the proposed `XWL` can still significantly improve the endurance of crossbar ReRAM memory even with longer address remapping intervals.

4.2.2 Performance Overhead. The data swapping after address remapping is inevitable for wear leveling, while it also contributes major performance overhead [6, 24]. We also evaluate the performance overhead of introducing the proposed wear leveling techniques. Figure 8 shows the swapping overhead in performance by using `Naïve` and `XWL` designs. The swapping overhead is defined as follows:

$$Swapping\ \ Overhead = \frac{t_{data\_swapping}}{t_{execution}} \quad (6)$$

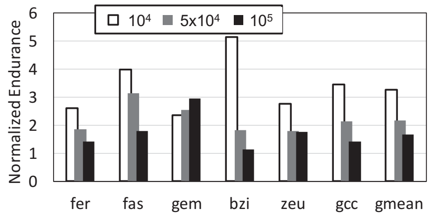


Figure 7: Comparison of normalized endurance with different remapping intervals.

where $t_{data\_swapping}$ and $t_{execution}$ represent total data swapping time and and execution time in cycles through whole memory system lifetime, which indicates the overall percentage of ReRAM crossbar lifetime are used for data migration. Overall, `Naïve` and `XWL` incur 6.5% and 6.1% performance overheads respectively. Though the `XWL` may potentially result in less hot ReRAM pages write to the rows with smaller RESET latency as well as a larger number of data swapping through the whole system lifetime, its performance loss is slightly better than `Naïve` since the `XWL` can much better improve the endurance cycles than `Naïve`.

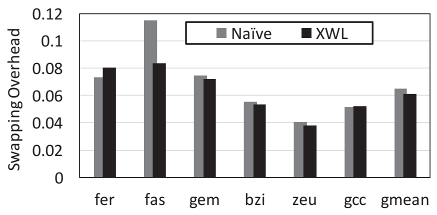


Figure 8: Comparison of data swapping overhead.

5 RELATED WORK

Wear leveling for non-volatile memories. Many prior work [14, 15] on enhancing PCM lifetime can apply to other resistive memories, and they shared the same general idea to evenly distribute write across all memory pages. Recent studies [24, 25, 25] on wear leveling for non-volatile memories took process variation (PV) issue into consideration, which leads that different page has non-uniform endurance. However, compared to this work, they all ignored the impact of array structures on write endurance, and fail to exploit the intrinsic features in ReRAM crossbars.

Crossbar resistive memory. The crossbar ReRAM architecture has recently attracted much attention [16, 18, 21, 22] owing to its smallest $4F^2$ planar cell size. In addition, due to its intrinsic analogy current accumulation feature, the crossbar resistive memory is also adopted to accelerate dot-product operation based convolutional neural network computations [4, 17]. Similar to crossbar ReRAM memory, the dot-product operation accelerators also suffer from limited write endurance when programming cells. Therefore, this work is critically important to crossbar resistive memory design as well as in-memory computing.

RESET latency discrepancy in ReRAM crossbar. Liang et al. [11] explored the correlation between data storage patterns and voltage drop in crossbar resistive memory without cell selectors. Zhang et al. [22] observed and leveraged the RESET latency discrepancy caused by row physical distance from write drivers to improve write performance. Wen et al. [18] presented that, in addition to row address impact, the bitline data patterns also lead to RESET latency discrepancy in ReRAM crossbar.

6 CONCLUSION

In this paper, we focus on mitigating the write endurance degradation from IR drop by proposing a novel wear leveling scheme for crossbar ReRAM memory. Specifically, we study the write endurance variation issue in crossbar ReRAM memory, and observe that the effective write, which indicates actual the degree of ReRAM wearing out, dynamically changes in runtime with different data patterns and row addresses. We propose a novel wear leveling scheme based on effective write to enhance lifetime of crossbar ReRAM memory. To the best of our knowledge, this paper is the first study specifically on addressing the write endurance issue for crossbar ReRAM memory. The final evaluation results reveal that, our design improves write endurance by 324%, compared to the baseline design.

ACKNOWLEDGMENTS

This research is supported in part by NSF CCF-1617071.

REFERENCES

[1] K. Albayraktaroglu, *et al.* Biobench: A benchmark suite of bioinformatics applications. In *ISPASS*, 2005.
[2] C. Bienia, *et al.* Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, volume 2011, 2009.
[3] Y. Chen, *et al.* Balancing set/reset pulse for $> 10^{10}$ endurance in HfO_2/Hf 1t1r bipolar rram. In *IEEE Trans. on Electron devices*, 2012.
[4] P. Chi, *et al.* Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA*, 2016.
[5] S. Cho, *et al.* Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *MICRO*, 2009.
[6] J. Dong, *et al.* Wear rate leveling: Lifetime enhancement of pram with endurance variation. In *DAC*, 2011.
[7] J. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.
[8] P. Huang, *et al.* Analytic model of endurance degradation and its practical applications for operation scheme optimization in metal oxide based rram. In *IEDM*, 2013.
[9] Z. Jiang, *et al.* Verilog-a compact model for oxide-based resistive random access memory (rram). In *SISPAD*, 2014.
[10] U. Kang, *et al.* Co-architecting controllers and dram to enhance dram process scaling. In *The memory forum*, 2014.
[11] J. Liang and *et al.* Cross-point memory array without cell selectors-device characteristics and data storage pattern dependencies. *IEEE Trans. on Electron Devices*, 2010.
[12] C. Luk, *et al.* Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.
[13] C. Nail, *et al.* Understanding rram endurance, retention and window margin trade-off using experimental results and simulations. In *IEDM*, 2016.
[14] M. Qureshi, *et al.* Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO*, 2009.
[15] N. Seong, *et al.* Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA*, 2010.
[16] M. Shevgoor, *et al.* Improving memristor memory with sneak current sharing. In *ICCD*, 2015.
[17] L. Song, *et al.* Pipelayer: A pipelined reram-based accelerator for deep learning. In *HPCA*, 2017.
[18] W. Wen, *et al.* Speeding up crossbar resistive memory by exploiting in-memory data patterns. In *ICCAD*, 2017.
[19] H. Wong, *et al.* Metal–oxide rram. *Proceedings of the IEEE*, 2012.
[20] H. Wu, *et al.* Resistive random access memory for future information processing system. *Proceedings of the IEEE*, 2017.
[21] C. Xu, *et al.* Overcoming the challenges of crossbar resistive memory architectures. In *HCPA*, 2015.
[22] H. Zhang, *et al.* Leader: Accelerating reram-based main memory by leveraging access latency discrepancy in crossbar arrays. In *DATE*, 2016.
[23] L. Zhang, *et al.* Mellow writes: Extending lifetime in resistive memories through selective slow write backs. In *ISCA*, 2016.
[24] X. Zhang *et al.* Toss-up wear leveling: Protecting phase-change memories from inconsistent write patterns. In *DAC*, 2017.
[25] M. Zhao, *et al.* Slc-enabled wear leveling for mlc pcm considering process variation. In *DAC*, 2014.
[26] N. Muralimanohar, *et al.* CACTI 6.0: A tool to model large caches In *HP Laboratories*, 2009.