CLIMBER: Defending Phase Change Memory Against Inconsistent Write Attacks

Zhuohui Duan, Haobo Wang, Haikun Liu, Xiaofei Liao, Hai Jin, Yu Zhang, Fubing Mao
National Engineering Research Center for Big Data Technology and System,
Services Computing Technology and System Lab, Cluster and Grid Computing Lab,
School of Computing Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
{zhduan,haobowang,hkliu,xfliao,hjin,zhyu,fbmao}@hust.edu.cn

ABSTRACT

Non-volatile Memories (NVMs) usually demonstrate vast endurance variation due to *Process Variation* (PV). They are vulnerable to an *Inconsistent Write Attack* (IWA) which reverses the write intensity distribution in two adjacent wear leveling windows. In this paper, we propose CLIMBER, a defense mechanism to neutralize IWA for NVMs. CLIMBER dynamically changes harmful address mappings so that intensive writes to weak cells are still redirected to strong cells. CLIMBER also conceals weak NVM cells from attackers by randomly mapping cold addresses to weak NVM regions. Experimental results show that CLIMBER can reduce maximum page wear rate by 43.2% compared with the state-of-the-art *Toss-up Wear Leveling* and prolong NVM lifetime from 4.19 years to 7.37 years with trivial performance/hardware overhead.

CCS CONCEPTS

• **Security and privacy → Hardware attacks and countermeasures**.

KEYWORDS

Non-volatile memory, Wear-out attack, Wear leveling

ACM Reference Format:
Zhuohui Duan, Haobo Wang, Haikun Liu, Xiaofei Liao, Hai Jin, Yu Zhang, Fubing Mao. 2022. CLIMBER: Defending Phase Change Memory Against Inconsistent Write Attacks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530546

1 INTRODUCTION

Non-volatile Memory (NVM) such as *Phase Change Memory* (PCM) can significantly enlarge main memory capacity in a cost and energy-efficient manner. However, PCM cells can only sustain a maximum 10^7–10^8 times of writes [1] before a permanent failure. Previous studies have shown that non-uniform writes to different PCM cells can significantly shorten the lifetime of PCM by 20× [2]. Wear leveling techniques [1–3] have been widely exploited to distribute writes uniformly throughout all NVM address space. They are often achieved by remapping intensive writes to less frequently-written NVM lines in memory controllers.

On the other hand, PCM cells often demonstrate huge endurance variation due to immature fabrication process [4, 5]. A recent study has shown that the number of writes endured by the weakest cell is 56X less than that of the strongest cell, making the lifetime of PCM have only 4.1% of the ideal lifetime when writes are uniformly distributed on all PCM cells [6]. Accordingly, PV-aware wear leveling schemes [4, 5, 7, 8] are widely studied, such as bloom filter-based dynamic wear leveling [8] and *Wear Rate Leveling* (WRL) [7]. The basic idea of these schemes is to redirect predicted intensive writes to strong cells so that all NVM cells have the same wear rate.

PV-aware wear leveling schemes usually contain three phases: prediction, swapping, and running [7, 9]. In the prediction phase, hot/cold memory addresses are predicted according to write traffic in the prediction phase. In the swapping phase, hot/cold memory addresses are remapped to strong/weak NVM cells in an address mapping table. In the running phase, if the write intensity in the current interval is consistent with the prediction, intensive writes are redirected to strong cells.

Unfortunately, the assumption that the future writes intensity distribution is consistent with the written pattern in the current observation window, is not always correct. For example, a malicious program may cheat PV-aware wear leveling schemes and allow the distribution of write intensity in running phase absolutely reverse to the prediction. In this way, the intensive writes are actually redirected to weak cells. Thus, malicious programs can exploit the process variation feature and PV-aware wear leveling schemes to accelerate the wear-out of weak NVM cells. This attack paradigm is called *Inconsistent Write Attack* (IWA) [9].

Figure 1 illustrates the workflow of the IWA, which is designed to mislead the address remapping during the swapping phase. Before the swapping phase, a malicious program generates non-uniform write traffic so that there are the hottest address V_a and the coldest address V_c in the memory. During the swapping phase, the PV-aware wear-leveling scheme maps the hottest address V_a to the strongest cell C_b and maps the coldest address V_c to the weakest cell C_d. Because the data swapping conducted by the IMC leads to a long memory response time, the malicious program can detect the swapping phase according to the observed high memory access latency [9]. After the swapping phase, the malicious program can flip the write traffic and generate an absolutely reverse distribution of write intensity to the same address. For example, the coldest

Zhuohui Duan and Haobo Wang contributed equally to this work. Haikun Liu is the corresponding author. This work is supported by National Natural Science Foundation of China under grants No.62072198, 61732010, 61825202, 62032008.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530546

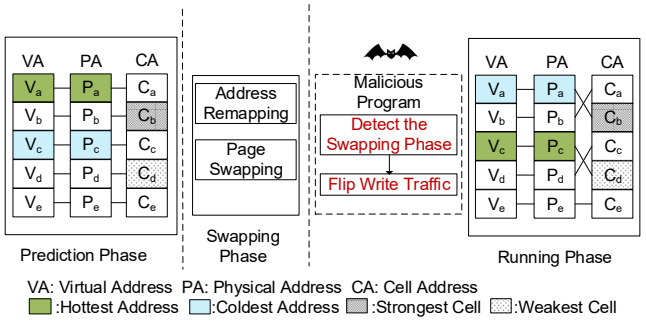

Figure 1: An illustration of inconsistent write attack

address V_c in the prediction phase becomes the hottest address in the running phase. As V_c is actually mapped to the weakest cell C_d, it can be worn out quickly by the intensive write traffic.

Recently, *Toss-up Wear Leveling* (TWL) [9] has been proposed to defend against this attack. TWL randomly distributes writes to two bonded memory blocks including a strong page A and a weak page B. For a write, it is allocated to page A with a probability of $\frac{Endurance_A}{Endurance_A+Endurance_B}$, without considering the hotness of memory addresses (i.e., write intensity). TWL sets a trigger interval for this operation to reduce page swapping times. However, TWL scheme causes a large amount of *negative page swapping* which even speeds up the wear-out of NVM cells (Section 2).

To counteract IWA, we propose a novel wear leveling scheme called CLIMBER. It dynamically changes harmful address mappings in which heavily written addresses are mapped to weak cells in the running phase. Thus, CLIMBER can correct prediction errors of wear-leveling algorithms and redirect intensive writes to strong cells based on a threshold of page hotness. Moreover, we propose a *Weak Page Randomly Mapping* (WPRM) scheme to protect the weakest NVM cell from IWA. CLIMBER randomly maps cold pages to a weak NVM region containing many weak pages, and thus an attacker cannot deliver intensive write traffic to a certain weak page. In this way, WPRM can spread intensive writes to a large number of weak pages and thus mitigate the effect of IWA. CLIMBER can be easily integrated into previous PV-aware wear leveling schemes to defend against IWA and further improve the lifetime of PCM.

We evaluate CLIMBER using memory trace collected by Zsim simulator [10]. The source code and memory trace of CLIMBER are available at [11]. Experimental results show that CLIMBER can defend NVMs against IWA efficiently. Compared with the state-of-the-art TWL [9], it can reduce the maximum and average page wear rate by 43.2% and 14.2%, respectively, and significantly reduce write amplification due to wear leveling by 82%. CLIMBER can prolong the lifetime of PCM from 4.19 years to 7.37 years with trivial performance and hardware overhead.

The rest of this paper is organized as follows. Section 2 presents the motivations of CLIMBER. Section 3 elaborates the defense mechanisms in CLIMBER. Section 4 presents experimental results. Section 5 introduces related work and we conclude in Section 6.

2 BACKGROUND AND MOTIVATION

PV-aware wear leveling schemes have been widely studied to mitigate the wear-out problem of NVMs. Most of those schemes should monitor write counts to identify hot addresses, and then predict the future write intensity distribution based on historical write patterns. Unfortunately, those wear leveling schemes are vulnerable to IWA.

Although the probability-based wear leveling scheme–TWL [9] is immune to IWA, it causes a large amount of *negative page swapping* which even speeds up the wear-out of NVM cells. In the following, we analyze the defect of TWL experimentally.

Table 1: Benchmarks for evaluation

Benchmarks	Applications
PBBS	BFS, DICT, ISORT, Matching, MIS, KNN, setCover, SpMV, MST
SPEC CPU2006	cactusADM, mcf, soplex
Parsec	freqmine, canneal, fluidanimate
Others	HPCC (GUPS), Linpack

We collect memory traces generated by several applications from *HPC Challenge Benchmark* (HPCC) [12], Linpack [13], SPEC CPU 2006 [14], *Problem Based Benchmarks Suit* (PBBS) [15], and Parsec [16], as shown in Table 1. Each application executes 40 billion instructions in Zsim simulator [10]. We evaluate the cost and effectiveness of TWL for different applications without IWA.

We implement TWL in a simulated system composed of 4GB PCM-based main memory [9]. TWL divides all PCM cells into multiple *toss-up pairs*, and performs a page swapping decision when a page of *toss-up pairs* is written more than 32 times. TWL chooses a page in *toss-up pairs* based on a probability model to serve the future write operations. Moreover, TWL also swaps a page with another randomly-chosen page when the page has been written 128 times. For an application whose memory footprint exceeds 4GB, we use a modulo operation to distribute the out-of-range addresses within the 4GB PCM. We divide the 4GB PCM into 1024 blocks. The endurance distribution of PCM cells is set according to [6]. The endurance of cells E can be calculated by Equation $E = 10^8 * (I^2 * R * T)^{-6}$, where I is the programming current, R is the resistance, and T is the write pulse width. We set the value of $R * T$ to 10.17 according to experimental results of previous studies [4–6, 17]. We generate 2048 current values which follow a Gauss distribution with a median value of 0.3 and a standard deviation of 0.033 [6]. We find that the endurance of the strongest cell is 25 times higher than that of the weakest cell. We assume 4KB pages in the same block have the same initial endurance.

Wear leveling schemes are often performed periodically. In each interval, we record the write count for each page in a counter. If the write count of the weak page is larger than that of the strong page in the current interval, but the write traffic is remapped from the strong page to the weak page in the last interval, we deem the page swapping is negative. On the other hand, if the write traffic on a weak page is higher than the write traffic on a strong page while a pair of weak and strong pages are not swapped in this interval, we deem that a *positive page swapping* is missing.

Figure 2 shows the proportion of *negative page swapping* caused by TWL. For all benchmarks, the percent of *negative page swapping* is about 36.8% on average. This implies that TWL introduces more write traffic to PCM and may degrade application performance due to *negative page swapping*.

Negative page swapping and missed *positive page swapping* both aggravate the wear-out of weak pages. We define *negative write traffic* as additional write traffic written to weak cells caused by *negative page swapping* and missed *positive page swapping*. As shown

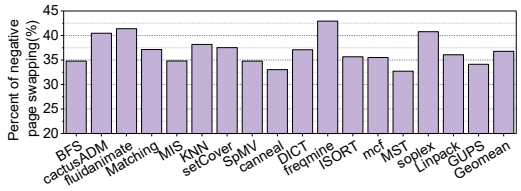

Figure 2: Negative page swapping due to TWL

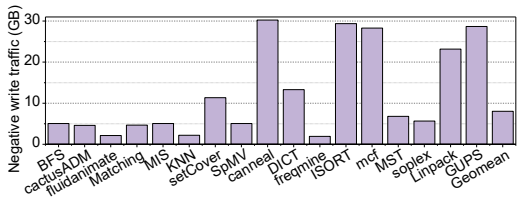

Figure 3: Negative write traffic due to negative page swapping and missed positive page swapping

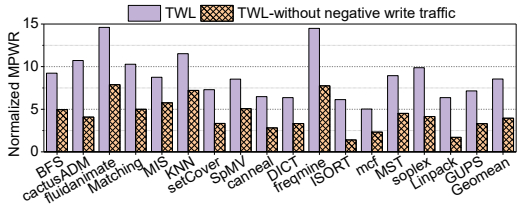

Figure 4: Normalized *maximum page wear rates* (MPWR) of TWL, all relative to the oracle page wear rate in an ideal case

in Figure 3, weak pages suffer 1.9 GB to 30.2 GB additional write traffic for different memory traces when running memory access trace 10 times. Note that each write leads to 64B write amplification according to the feature of PCM. The *negative write traffic* approximates to the memory footprint of each application, resulting in significant wear-out of NVM and memory bandwidth consumption.

To quantify the effectiveness of TWL on wear leveling, we leverage a widely-used metric–*Maximum Page Wear Rate* (MPWR), i.e., the maximum value of $\frac{write-traffic}{write-endurance}$ for all pages. We maintain an endurance table to record the endurance of each page for TWL. An endurance table entry is updated when a page is written. We can calculate MPWR at the end of each interval. We assume an ideal case in which an oracular wear leveling scheme can guarantee all pages have the same (oracle) wear rate. We can also calculate the MPWR in TWL assuming there is no *negative write traffic* in each interval. Figure 4 shows the MPWR in TWL and *TWL-without negative write traffic* are about 8.55 and 3.95 times higher than the oracle page wear rate on average. This clearly indicates that TWL is not the optimal solution for PV-aware wear leveling.

The above observations and findings motivate us to design a more effective and robust PV-aware wear leveling scheme to counteract IWA. We revisit address hotness-based PV-aware wear leveling schemes and explore new mechanisms to defend against IWA.

3 DESIGN AND IMPLEMENTATION

3.1 PV-Aware Wear Leveling

In previous hotness-based PV-aware wear leveling schemes, the address mappings are not changed until the next swapping phase. If the distribution of write intensity changes during the running phase, the wear leveling schemes become invalid and may even

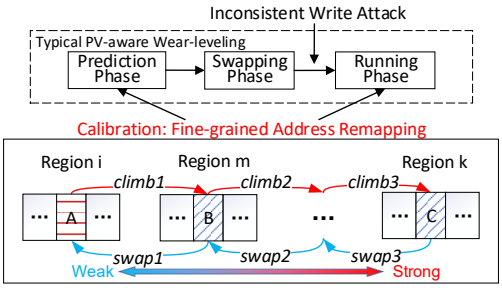

Figure 5: An illustration of wear leveling in CLIMBER

speed up the wear-out of NVM cells. Thus, those schemes are vulnerable to IWA. We propose CLIMBER, a novel PV-aware wear leveling scheme to address this problem. CLIMBER can dynamically change harmful address mappings in which hot addresses are mapped to weak cells. CLIMBER detects heavily-written pages in the memory controller and remaps hot pages to strong NVM regions progressively by using a hill-climbing algorithm. In this way, CLIMBER can correct prediction errors of wear-leveling schemes in a fine-grained manner during the prediction and running phases.

As shown in Figure 5, CLIMBER divides the memory space into n regions ordered by their endurance. Assume region 0 and region $n-1$ are the weakest and strongest regions, respectively. At first, page A is mapped to region i. If the write counts of page A is several times larger than a given threshold called *climb-threshold*, a stronger region m is randomly selected, and page B in the region m is also selected randomly. If the write counts and wear rate of A are both larger than that of B, and vice versa, we swap page A and page B. If the write traffic on page A is still much larger than the *climb-threshold* after the page swapping, it will eventually be swapped to the strongest region via continuously climbing. If the write counts of a page in the strongest region exceed its threshold, the page is swapped with a random-selected page in the same region. CLIMBER also uses a register p to choose a strong region s ($n/2 \le s \le n-1$) as the strongest region in a descending order. CLIMBER changes the value of p during each swapping phase to achieve balanced wear rate among these strong regions.

The value of *climb-threshold* for each page is set by considering the page endurance and the cost of page swapping. If the endurance of the weakest page is E_0 and its *climb-threshold* is T_0, we set the *climb-threshold* of a page with endurance E_1 as $\lfloor \frac{T_0 * E_1}{E_0} \rfloor$. In this way, we set a small threshold for a weak page to deliver write traffic to other pages immediately, and a large threshold for a strong page to mitigate the frequency of page swapping.

CLIMBER can leverage existing access counters of hotness-based PV-aware wear leveling schemes and cause limited additional storage overhead. Moreover, the deliberate *climb-threshold* can also limit the cost of additional page swapping. CLIMBER is complementary to existing PV-aware wear leveling schemes to counteract IWA and further reduces the maximum page wear rate.

3.2 Weak Pages Randomly Mapping

To decrease the maximum page wear rate, previous PV-aware wear-leveling schemes such as HC-to-SW and BWL map hot/cold pages to strong/weak regions in a fixed order. This allows attackers to identify and attack the weakest page easily. To conceal the addresses of

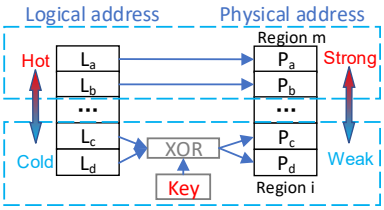

Figure 6: An illustration of weak pages randomly mapping

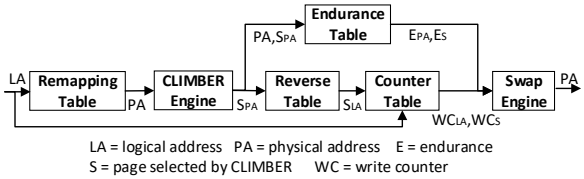

Figure 7: Data flow of write operations in CLIMBER

weakest pages, we design a *weak pages randomly mapping* (WPRM) scheme to randomly map cold addresses to weak regions whose endurance is lower than the median.

We illustrate WPRM in Figure 6. In each swapping phase, CLIMBER generates a random key for WPRM. According to previous PV-aware wear-leveling schemes, cold logical addresses L_c and L_d should be mapped to the weakest physical addresses P_c and P_d in the weakest region i. To conceal these address mappings, CLIMBER performs a *XOR* operation between the original physical addresses selected by previous PV-aware wear-leveling schemes and the random *key*, and then updates the remapping table using the new physical addresses for L_c and L_d. In this way, CLIMBER can randomly map cold logical addresses in a large weak region, and thus attacker cannot deliver malicious write traffic to the weakest page at all times, mitigating the effect of IWA. For the hot addresses L_a and L_b, which are mapped to P_a and P_b, WPRM does not change these mappings. As WPRM is only performed in the swapping phase and programs still rely on the address remapping table to access data in PCM cells directly, it causes negligible runtime overhead.

3.3 Data Flow of Read/Write Operations

In CLIMBER, read operations are performed like other wear leveling schemes. The address mapping table is first consulted to get the *Physical Address* (PA), and then data is read. For write operations, the data flow in CLIMBER is shown in Figure 7. CLIMBER uses the *Logical Address* (LA) to retrieve the PA from the *Remapping Table*. If the address should be remapped to a strong page, a page (S_{PA}) in a stronger region is selected by CLIMBER according to the endurance rank maintained in the *CLIMBER Engine*, which contains tables to record the climb-threshold, region number, and endurance rank. Because the *Write Counter* (WC) Table is indexed by logic addresses, we use a *Reverse Table* to maintain physical-to-logical address mappings. We use S_{PA} to retrieve the selected logical address (S_{LA}) from the Reverse Table, and then get the write counts of S_{LA} from the WC table. The page endurance of PA and S_{PA} are retrieved from the *Endurance Table*, which contains a table to record the original sustainable write counts, and another table to record the remaining sustainable write counts. At last, taking WC_{LA}, WC_S, E_{PA}, and E_S as inputs, the *Swap Engine* decides whether pages PA and S_{PA} should be swapped.

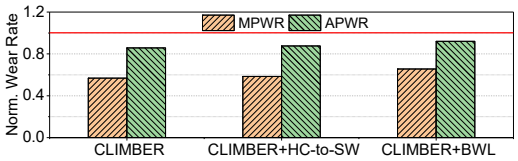

Figure 8: MPWR and APWR of CLIMBER under IWA, all relative to TWL

4 EVALUATION

In this section, we evaluate the effectiveness and efficiency of CLIMBER. We simulate a 4GB PCM main memory and maintain the lifetime of memory blocks at the granularity of 4KB pages. The lifetime of pages is described in Section 2. We use application memory traces described in Section 2 to generate write traffic on PCM.

4.1 Effectiveness of CLIMBER

Similar to previous works [7–9, 17], we use *maximum page wear rate* (MPWR) to evaluate the effectiveness of wear leveling and defense schemes in CLIMBER. We also use *average page wear rate* (APWR) as a new metric to evaluate CLIMBER. A good wear leveling scheme should minimize both MPWR and APWR. We also integrate CLIMBER with two other hotness-based wear leveling schemes—HC-to-SW [7] and BWL [8]. HC-to-SW maps hot/cold addresses to strong/weak pages to balance the wear rate of all pages. BWL has a similar idea with HC-to-SW, but further exploits bloom filters to decrease the storage overhead of write counters. These two schemes are both vulnerable to IWA [9]. We compare CLIMBER with *Toss-up Wear Leveling* (TWL) [9].

Figure 8 shows the MPWR and APWR of CLIMBER under IWA, all relative to TWL. CLIMBER can reduce MPWR and APWR by 43.2% and 14.2%, respectively. Because TWL may incorrectly place intensive write traffic on weak pages due to randomly page swapping, the page with the MPWR often located in a weak region. In contrast, CLIMBER can place intensive write traffic in strong regions, the page with the MPWR appears in a strong region. Since CLIMBER can reduce negative page swapping, it achieves lower MPWR than TWL. For the same write traffic, weak pages can be worn out faster than strong pages. Thus, the page wear rate is more sensitive to writes on weak pages. Correspondingly, APWR increases faster if intensive writes are mapped to weak pages. Because CLIMBER always delivers intensive write traffic to strong pages, it achieves a lower APWR than TWL.

Moreover, CLIMBER reduces the MPWR of HC-to-SW and BWL by 41.5% and 34.4% compared with TWL, respectively. The APWR is also reduced slightly. Although malicious programs can deliver intensive writes to weak pages, CLIMBER can redirect the write traffic to strong regions again, and the WPRM strategy also conceals the weakest page from malicious programs after each swapping phase. Thus, CLIMBER can defend the traditional wear leveling algorithms against IWA.

We also evaluate the effectiveness of CLIMBER for different benchmarks when they do not suffer IWA. As shown in Figure 9 and Figure 10, compared with TWL, CLIMBER can reduce MPWR and APWR by 55.4% and 30.3% on average, respectively. CLIMBER reduces MPWR of HC-to-SW and BWL by 53.8% and 53.3%, respectively. APWR also drops significantly compared with TWL.

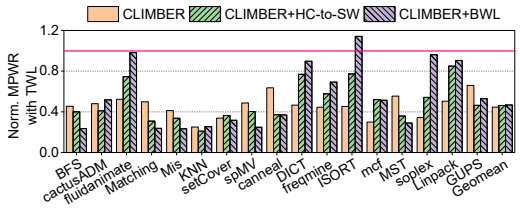

Figure 9: Normalized MPWR of CLIMBER without IWA, all relative to TWL

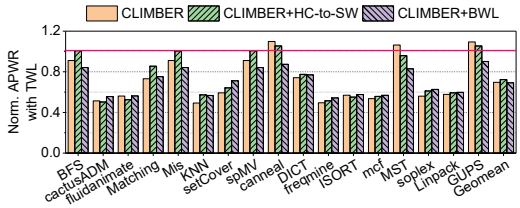

Figure 10: Normalized APWR of CLIMBER without IWA, all relative to TWL

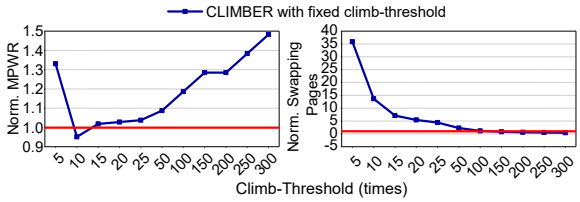

Figure 11: Normalized MPWR and page swapping overhead of CLIMBER with a fixed climb-threshold, all relative to CLIMBER with a variable climb-threshold

Table 2: Lifetime and page swapping under IWA

Wear-leveling schemes	Page swapping (GB)	Lifetime (year)
TWL	**58.3**	**4.19**
CLIMBER	**10.5 (82%↓)**	**7.37 (75.9%↑)**
CLIMBER+HC-to-SW	**10.4 (82.1%↓)**	**7.17 (71.0%↑)**
CLIMBER+BWL	**19.6 (66.5%↓)**	**6.39 (52.5%↑)**

In the following, we evaluate the lifetime of PCM for different wear leveling schemes under IWA. Assuming a malicious program continuously writes data to PCM following the IWA model, we can calculate the lifetime of PCM via $lifetime = T/B$, where T represents the maximum write traffic that the PCM device can endure, and B represents write traffic per second. T can be estimated by $\frac{T'}{MPWR}$, where T' and $MPWR$ represent the total write traffic and the *maximum page wear rate* in our experiments, respectively. It is well-known that the burst write mode only allows programs to write at most 64B (size of a cache line) to main memory once. The latency of each write operation is about 250 ns [9]. Thus, B can be estimated as $64B/250ns$. Finally, we can calculate the lifetime of PCM under IWA for all wear leveling schemes. As shown in Table 2, CLIMBER can prolong the lifetime of PCM from 4.19 years to 7.37 years compared with TWL.

Figure 11 shows the MPWR and the numbers of page swapping for different CLIMBER derivations with a fixed climb-threshold, all relative to CLIMBER with variable climb-threshold in our strategy (the red line, i.e., 1.0). As shown in Figure 11, CLIMBER using variable climb-threshold achieves a low MPWR and much less overhead of page swapping at the same time compared with the fixed climb-threshold strategy. Although CLIMBER using a fixed climb-threshold of 10 achieves a lower MPWR than the variable climb-threshold strategy, it causes a large amount of page swapping.

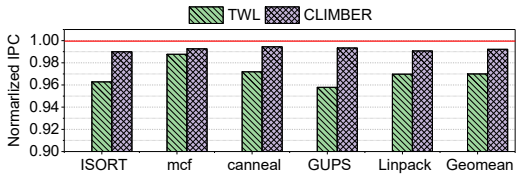

Figure 12: Normalized IPC for different wear-leveling schemes, all relative to the normal executions without any wear-leveling scheme

4.2 Overhead

We first evaluate the hardware overhead of CLIMBER. As shown in Figure 7, for each 4KB page, CLIMBER should maintain an 8-bit entry for the climb-threshold, a 32-bit entry in the endurance table to record the original write counts that a page can sustain, a 32-bit entry to record the remaining sustainable write counts in another table, a 3-bit entry to record the region number, a 20-bit entry to record the endurance rank, a 20-bit remapping table entry, a 20-bit entry to record the logical page number in the reverse table, and a 20-bit entry in the write counter table to record the writes to the page. Totally, one 4KB page requires 155 bits to maintain the information for our wear leveling strategy. Thus, the storage overhead is 155bits/4KB = 0.47%.

We also evaluate the additional write traffic of page swapping introduced by CLIMBER, as shown in Table 2. Compared with TWL, CLIMBER significantly reduces negative page swapping and leads to less additional write traffic because the distribution of write intensity is considered in page swapping. Under the IWA, TWL introduces 58.3 GB additional write traffic. CLIMBER can reduce the overhead by 82% compared with TWL. CLIMBER integrated with HC-to-SW further reduces the additional write traffic to 10.4 GB, a reduction of 82.1% compared with TWL.

To evaluate the performance overhead of CLIMBER due to page swapping, we run applications with high write traffic in Zsim simulator [10, 18]. Figure 12 shows application *instructions per cycle* (IPC), all relative to the executions without any wear leveling scheme (baseline). TWL randomly swaps pages in a toss-up pair, and thus causes 3% performance overhead on average compared with the baseline. CLIMBER leads to only 0.78% performance penalty due to page swapping because we consider both page hotness and endurance to determine the threshold of page swapping, and only swap pages whose hotness exceeds the given threshold.

4.3 Sensitivity Studies

The effectiveness of CLIMBER is mainly affected by two parameters: the region size and *climb-threshold*. In the following, we conduct sensitivity studies on these two parameters. Figure 13(a) shows how the region size affects MPWR and APWR. The number i in X-axis represents the power of 2 (2^i) pages within a region. When the region size decreases, CLIMBER may wear out the strong region first. Thus, MPWR roughly decreases with the increase of region size. When the region size becomes very large (e.g., 2^{19} pages), CLIMBER actually maps hot addresses to weak regions randomly, resulting in an increase of MPWR for weak cells.

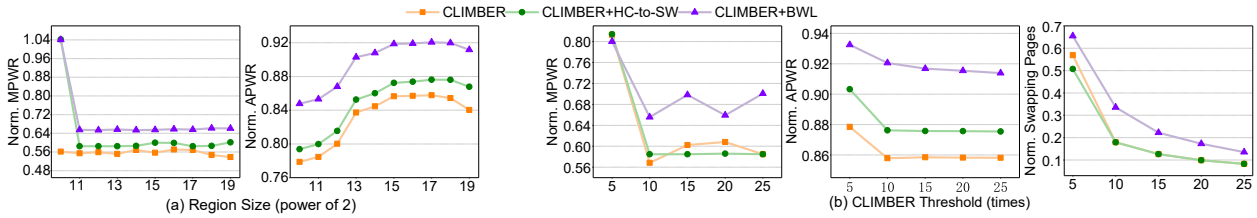

Figure 13: The impact of region sizes and *climb-thresholds* on MPWR, APWR, and page swapping cost, all relative to TWL

Since CLIMBER selects strong regions in a descending order for wear leveling, when the region size increases, more weak pages are selected by CLIMBER to serve write traffic. Thus, APWR shows an upward trend when the region size is less than 2^{15} pages in our experiments. When the region size is greater than 2^{15} pages, CLIMBER delivers write traffic to the top 50% strong regions evenly, and thus APWR tends to decline slightly because writes on strong pages have less impact on APWR.

Climb-threshold also has a non-trivial impact on MPWR, APWR, and write traffic due to page swapping, as shown in Figure 13(b). The number i in the X-axis represents that the weakest page should be swapped to a stronger page after it is written i times. Under IWA, CLIMBER can remap hot pages to strong regions quickly, and thus the MPWR usually occurs in the strong regions. A lower *climb-threshold* usually leads to frequent page swapping which increases the wear of NVM pages. Thus, both MPWR and APWR decrease with the growth of *climb-threshold*.

Based on our experimental results, to balance the effectiveness and overhead of CLIMBER, the optimal region size is 17, and the optimal *climb-threshold* for the weakest page is 10.

5 RELATED WORK

NVM attack and defense have aroused increasing interest recently. Huang et.al. [1] identify a focus-on write attack that exploits the asymmetry of write time (i.e., set and reset) of PCM to speculate the address mapping of the wear-leveling scheme. SecurityRBSG is then proposed to defend against this attack using a multi-level dynamic remapping mechanism. However, SecurityRBSG is designed for PCM devices without endurance variation, and thus its application scope is limited. CLIMBER is immune to this attack because it can change harmful address mappings. Xu et. al. [6] identify a *Uniform Address Attack* (UAA), which writes all memory rows uniformly to wear out weak rows earlier. They propose a spare-line replacement scheme called Max-WE to maximize the weak lines' endurance. Max-WE employs weak-priority and weak-strong-matching strategies to replace failed lines with spare lines in weak regions. Max-WE focuses on fault-tolerant and is complementary to CLIMBER to further prolong the lifetime of PCM when some PCM cells have been worn out. TWL [9] is proposed to defend PCM against IWA which uses inconsistent write patterns to mislead PV-aware wear leveling schemes. TWL advocates probability-based address remapping to replace address hotness-based wear leveling schemes. However, TWL results in significant write amplification due to negative page swapping. In contrast, CLIMBER can still exploit the address hotness monitoring mechanism to minimize the maximum page wear rate, and can also defend against IWA by correcting the harmful address mappings in a fine-grained manner.

6 CONCLUSION

NVMs are vulnerable to an inconsistent write attack which reverses the write intensity distribution in two adjacent wear leveling windows. In this paper, we present CLIMBER, a defense mechanism against IWA. CLIMBER leverages a novel wear leveling algorithm to dynamically changes harmful address mappings so that intensive writes to weak cells are redirected to strong cells. CLIMBER also conceals weak NVM cells from attackers by randomly mapping cold addresses to weak NVM regions. Experiment results show that CLIMBER can defend NVM against IWA efficiently with negligible performance and hardware overhead.

REFERENCES

[1] Fangting Huang, Dan Feng, Wen Xia, Wen Zhou, Yucheng Zhang, Min Fu, Chuntao Jiang, and Yukun Zhou. Security RBSG: Protecting Phase Change Memory with Security-Level Adjustable Dynamic Mapping. In *Proceedings of IPDPS*, 2016.

[2] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *Proceedings of MICRO*, 2009.

[3] Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-Change Memory with Dynamically Randomized Address Mapping. *ACM SIGARCH Computer Architecture News*, 38(3):383–394, 2010.

[4] Kinarn Kim and Su Jin Ahn. Reliability Investigations for Manufacturable High Density PRAM. In *Proceedings of IRPS*, 2005.

[5] Wangyuan Zhang and Tao Li. Characterizing and Mitigating the Impact of Process Variations on Phase Change Based Memory Systems. In *Proceedings of MICRO*, 2009.

[6] Jie Xu, Dan Feng, Yu Hua, Fangting Huang, Wen Zhou, Wei Tong, and Jingning Liu. An Efficient Spare-Line Replacement Scheme to Enhance NVM Security. In *Proceedings of DAC*, 2019.

[7] Jianbo Dong, Lei Zhang, Yinhe Han, Ying Wang, and Xiaowei Li. Wear Rate Leveling: Lifetime Enhancement of PRAM with Endurance Variation. In *Proceedings of DAC*, 2011.

[8] Joosung Yun, Sunggu Lee, and Sungjoo Yoo. Bloom Filter-based Dynamic Wear Leveling for Phase-Change RAM. In *Proceedings of DATE*, 2012.

[9] Xian Zhang and Guangyu Sun. Toss-up Wear Leveling: Protecting Phase-Change Memories from Inconsistent Write Patterns. In *Proceedings of DAC*, 2017.

[10] Daniel Sanchez and Christos Kozyrakis. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. *ACM SIGARCH Computer Architecture News*, 41(3):475–486, 2013.

[11] CLIMBER. https://github.com/CGCL-codes/CLIMBER.

[12] Piotr R. Luszczek, David H. Bailey, Jack J. Dongarra, Jeremy Kepner, Robert F. Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The HPC Challenge (HPCC) Benchmark Suite. In *Proceedings of SC*, 2006.

[13] Linpack. http://www.netlib.org/benchmark.

[14] SPECCPU2006. https://www.spec.org/cpu2006.

[15] Julian Shun, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Aapo Kyrola, Harsha Vardhan Simhadri, and Kanat Tangwongsan. Brief Announcement: The Problem Based Benchmark Suite. In *Proceedings of SPAA*, 2012.

[16] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[17] Wen Zhou, Dan Feng, Yu Hua, Jingning Liu, Fangting Huang, and Pengfei Zuo. Increasing Lifetime and Security of Phase-Change Memory with Endurance Variation. In *Proceedings of ICPADS*, 2016.

[18] Haikun Liu, Yujie Chen, Xiaofei Liao, Hai Jin, Bingsheng He, Long Zheng, and Rentong Guo. Hardware/Software Cooperative Caching for Hybrid DRAM/NVM Memory Architectures. In *Proceedings of ICS*, 2017.