Toss-up Wear Leveling: Protecting Phase-Change Memories from Inconsistent Write Patterns

Xian Zhang[†]
zhang.xian@pku.edu.cn

Guangyu Sun[†,‡]
gsun@pku.edu.cn

[†]Center for Energy-Efficient Computing and Applications, Peking University, Beijing 100871, China
[‡]Collaborative Innovation Center of High Performance Computing, NUDT, Changsha 410073, China

ABSTRACT

Limited write endurance is one of major obstacles to adopt Phase Change Memories (PCMs) in practice as future main memory. Considering process variation (PV) and non-uniform write intensity, PCM cells with low endurance (i.e. weak cells) can wear out in seconds under intensive writes. To prolong PCMs' lifetime, many PV-aware wear leveling schemes have been proposed following a common idea: intensive writes are predicted and allocated to cells with high endurance (i.e. strong cells) based on the write intensity distribution, which should be consistent at predicted intervals. However, we discover that this idea leaves a serious vulnerability against a malicious program, which is designed to have an inconsistent write intensity distribution. Prior wear-leveling schemes can even be leveraged to speed up wearing out weak cells. To counteract this attack, we propose Toss-up Wear Leveling (TWL), a novel scheme that randomly allocates writes between two bond blocks discounting the consistency of write distribution. Experiment results demonstrate that, compared to prior works, TWL can improve lifetime substantially with negligible overhead in performance and hardware cost.

1. INTRODUCTION

Phase Change Memories (PCMs) have been increasingly proposed as a competitive candidate for future main memories of servers [10, 12, 9, 16]. However, limited write endurance has impeded the adoption of PCMs in practice. A PCM cell is supposed to sustain only about 10^8 writes before a permanent failure occurs [1, 12]. To tackle the problem, many wear leveling schemes are proposed to uniformly distribute writes [10, 12, 7].

Unfortunately, when process variation (PV) is considered, some cells may be inherently susceptible to writes, which are called weak cells. And traditional wear leveling, which leads to an even write intensity, may reallocate writes to the weak cells to speedup their wear-out [6, 15, 13]. Therefore, PV-aware wear leveling are widely studied, such as wear-rate leveling, bloom-filter based leveling and etc [14, 6, 13, 15].

This work was supported by NSF China 61572045.

DAC'17, June 18-22, 2017, Austin, TX, USA
©2017 ACM. ISBN 978-1-4503-4927-7/17/06 ...$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062329.

A general flow of PV-aware wear leveling includes three phases: prediction, swap, running [6]. In the prediction phase, hot/cold addresses are figured out according to the write traffic. In the swap phase, hot addresses are mapped to strong cells and vice versa. In the running phase, if the distribution of write intensity is consistent, strong cells are intensively written while weak cells suffer a much lower write traffic. Therefore, weak cells are protected and PCM's lifetime is prolonged.

Unfortunately, the bedrock of these techniques, namely the consistency of write intensity distribution before and after swaps, may be incorrect for malicious programs. For a malicious program, in order to wear-out the PCM, it may cheat in the prediction phase and show a reverse distribution of write intensity in the running phase. As a result, intensive writes are imposed on the weak cells resulting in a quick wear-out.

To mitigate the "inconsistent write" attack, we propose a novel PV-aware wear leveling, which does not rely on the consistency of write distribution but randomly reallocates the writes according to cells' endurance. The core idea of our design is straightforward: a strong page (page-A) is bond with a weak page (page-B) to be a "toss-up pair". Every time a write is to be allocated to any page in the pair, with a probability of $\frac{Endurance-A}{Endurance-A+Endurance-B}$, the write will be reallocated to page-A otherwise page-B. The process is like playing a **"toss-up"** to decide which page to write. The page with high endurance in a pair are likely to be written more frequently. Thus, our design is PV-aware and immune to the "inconsistent write" attack. Further optimizations called inter-pair swap, strong-weak paring and interval-triggered toss-up are also proposed.

Our contributions can be summarized as follows:

- We introduce a novel wear-out attack with inconsistent write patterns to circumvent existing PV-aware wear leveling schemes.
- We propose an efficient countermeasure named Toss-up Wear Leveling (TWL) to mitigate the attack .
- We further optimize TWL for better robustness and less write overhead.
- We evaluate TWL in the overhead of lifetime, performance and design overhead .

The rest of the paper is organized as follows: In Section 2, we introduce the related work of PV-aware wear leveling schemes. In Section 3, we present an effective attack to existing PV-aware wear leveling schemes. Section 4 introduces Toss-up wear leveling to counteract the inconsistent write attack. Comprehensive results are provided in Section 5, followed by our conclusion.

2. RELATED WORK

Due to the immature fabrication process and limitations of materials, the write endurance of a PCM cell is about 10^8, several order of magnitude lower than DRAM's [1, 12]. To prolong PCM's lifetime, traditional wear leveling schemes are widely studied, which aim to uniformly distribute writes among arrays [10, 12, 7].

Unfortunately, process variation (PV) can aggravate the wear-out problem and severely degrade the efficiency of traditional wear leveling schemes [14, 15, 5, 1, 13] . Some cells may tolerate much less writes [1, 14, 6] and therefore a uniform write distribution will speed up the wear-out of weak cells, which leads to a shorter PCM lifetime. Consequently, PV-aware wear leveling is proposed [15, 6, 13, 14, 1].

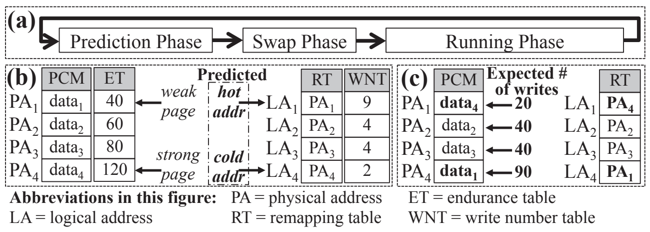

Figure 1: Illustration of a typical PV-aware wear leveling (Wear Rate Leveling): (a) The basic flow (b) states of PCM in the prediction phase (c) states of PCM in the running phase

Wear rate leveling [6] is used to illustrate the basic flow of PV-aware wear leveling schemes. As shown in Figure 1 (a), the basic flow includes three phases: prediction, swap and running. The initial endurance is supposed to be tested by the manufacturers [6, 14] and stored in the endurance table (ET). The mappings between logical address (LA) and physical address (PA) are recored in the remapping table (RT) [15, 14, 6]. In the prediction stage, the write number to every LA is recorded in the write number table (WNT) (Figure 1 (b)). If the write distribution is consistent in the next period, the future hot/cold addresses can be predicted. As shown in Figure 1 (b), LA_1 is predicted to be a hot address while LA_4 is a cold address. Then, by swapping the blocks, hot/cold addresses are reallocated to the strong/weak cells. As illustrated in Figure 1 (c), the LA_1 is mapped to PA_4 while LA_4 to PA_1. The last phase is the running phase, in which cells are written according to the updated RT. Note that running phase is much longer than the prediction phase (e.g. 10X in [6]). As shown in Figure 1 (c), endurance of $PA_i (i = 1, 2, 3, 4)$ is supposed to afford the expected number of writes.

Other PV-aware wear leveling schemes follow a similar flow. Yun *et. al.* further optimizes the flow using bloom filters and dynamic thresholds to identify hot/cold addresses and strong/weak cells [13]. Thus, the cycles of three phases may dynamically change and the sorting overhead in wear rate leveling is reduced. Zhao *et. al.* follow the same idea of Figure 1 considering the hybrid memory with MLC and SLC [15]. An OS-assisted scheme is proposed by Zhang *et. al.* to achieve the hot/cold to strong/weak mapping [14]. Asadinia *et. al.* have proposed a dynamic remapping scheme to slow the permanent failures related to process variation [1].

3. WEAR-OUT ATTACK BASED ON INCONSISTENT WRITE PATTERNS

In this section, we illustrate the details of our wear-out attack. We first describe the attack model. Then, we present our "inconsistent write attack".

3.1 Attack Model

We adopt an attack model similar to prior work [12, 7]. As shown in Figure 2, a PCM is employed as the main memory of a server. And the PCM is escorted by a PV-aware wear leveling scheme. The attacker can send malicious codes to compromise the OS remotely (e.g. via buffer overflow). And the CPU cache and DRAM buffer can be turned off by the OS. Arbitrary memory commands *(op,data,addr) tuple* can be sent to the PCM main memory. Here *op* is read or write. And *data* is the data to write (data is *null* when *op* is read). *addr* means the **logical address** to read or write. In addition, we assume that the attacker can use some instructions (e.g. $rdtsc()$) to measure the memory response time. The inner states within the PCM and the wear leveling circuits are not exposed to anyone.

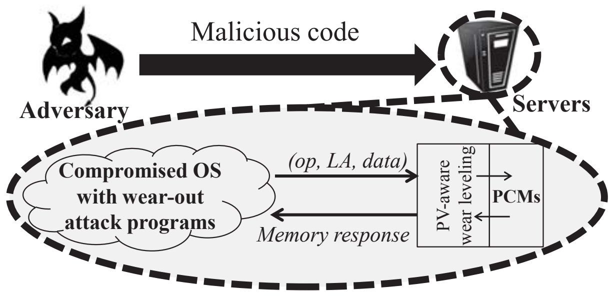

Figure 2: The attack model

3.2 Deliberate Inconsistent Write Patterns

As addressed before, prior PV-aware wear leveling schemes [14, 6, 13] share a basic assumption: write distribution is consistent. Thus, the hot/cold addresses of next running phase are determined according to the WNT during prediction phase. While the assumption is true for most normal benchmarks, it can be leveraged by a malicious program to speedup the wear-out of weak cells. The malicious program can quickly wear-out a memory page by repeating two steps as follows:

- Step-1: Write LA_i for W_i times $(i = 1, 2, ..., N)$. Ensure that $W_1 < W_k < W_N (k = 2, ..., N - 1)$. Meanwhile, keep detecting the start and end of swap phase by measuring the memory response time[1].

[1]Memory swaps will block all memory requests to ensure memory integrity, which leads to an increase in memory response time [7, 12].

- Step-2: When swap phase ends, write LA_i for W'_i times ($i = 1, 2, ..., N$) with $W'_N < W'_k < W'_1 (k = 2, ..., N-1)$. Detect the swap phase as in Step-1.

Step-1 is to mislead the prediction that LA_1 is a cold address. Meanwhile, the page at LA_N is heavily written. Thus, after the swap phase, LA_1 should be mapped to a weak page. In step-2, the malicious program imposes intensive writes to LA_1 which is a weak page. And LA_N plays the same role as LA_1 in step-1. When step-1 is executed again, the weak page corresponding to LA_N will be heavily written. Consequently, the weak page is always under intensive writes. It is like that PV-aware wear leveling "exposes" the weak page to intensive writes, which speeds up the wear-out of weak pages.

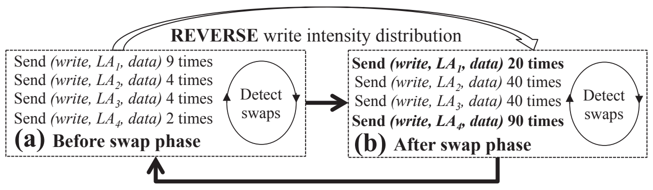
REVERSE write intensity distribution
Send (write, LA1, data) 9 times
Send (write, LA2, data) 4 times
Send (write, LA3, data) 4 times
Send (write, LA4, data) 2 times
Detect swaps
(a) Before swap phase
Send (write, LA1, data) 20 times
Send (write, LA2, data) 40 times
Send (write, LA3, data) 40 times
Send (write, LA4, data) 90 times
Detect swaps
(b) After swap phase

Figure 3: Inconsistent write attack towards prior PV-aware wear leveling schemes: (a) step-1 (b) step-2

Figure 3 illustrates a successful attack to wear rate leveling in Figure 1. The write distribution in Figure 3 (a) is the same with that in Figure 1 (b). After the swap phase, LA_4 is mapped to a weak page at PA_1 (Figure 1 (c)). Then, in Figure 3 (b), 90 times of writes will wear-out the page at PA_1.

The attack still works when other PV-aware wear leveling schemes (e.g. bloom-filter based wear leveling [13]) are adopted because of two reasons: (1) our attack does not rely on the fixed length of prediction phase or running phase, and (2) the write number to every page can be properly set to make LA_N or LA_1 detected as cold addresses by the bloom filters.

4. TOSS-UP WEAR LEVELING

In this section, we propose a novel wear-leveling scheme called Toss-up wear leveling (TWL) to counteract inconsistent writes. First, we present the core idea of TWL. Then, we establish a simple mathematical model of TWL's write overhead. Based on the model, we propose two optimizations to lower write overhead. Last, we present the overview of TWL read/write flows.

4.1 The Basic Idea of TWL

According to the last section, it is dangerous to rely on the consistency of write distribution when malicious program exploits an inconsistent write pattern. Therefore, we propose a PV-aware wear leveling scheme NOT based on the prediction of the future write distribution, but based on a more intrinsic idea of PV-aware wear leveling: *the stronger the page, the more writes it should undertake.*

As shown in Figure 4 (a), we assume that page-A and page-B are bond as a **toss-up pair**. And page-A's endurance is E_A while page-B's is E_B. If there is a write toward page-A or page-B (i.e. $Addr_{write} = Addr_A$ or

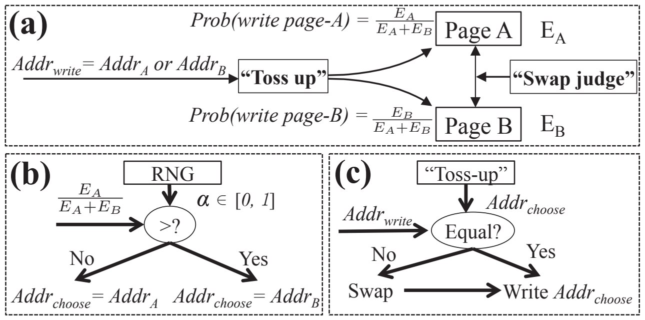

Figure 4: The core idea of TWL: (a) the overview (b) illustration of "toss-up" (c) illustration of "swap judge"

$Addr_B$), the core idea of TWL is to randomly reallocate the write to page-A or page-B with the possibility of $\frac{E_A}{E_A+E_B}$ and $\frac{E_B}{E_A+E_B}$, respectively. This process is like playing a "toss-up" and ensures that stronger pages are allocated more writes, without relying on the prediction of write intensity distribution.

To implement the core idea, two components are required. The first one is called "toss-up", which is shown in Figure 4 (b). We employ a random number generator (RNG) which can generate a random number $\alpha \in [0, 1]$. Then, the random number is compared with $\frac{E_A}{E_A+E_B}$ to decide which address (denoted as $Addr_{choose}$) to undertake the write. When $\alpha < \frac{E_A}{E_A+E_B}$, page-A is chosen otherwise page-B. However, when $Addr_{choose}$ differs from $Addr_{write}$, the data stored at $Addr_{choose}$ previously will be lost if we directly write $Addr_{choose}$ with data of $Addr_{write}$.

To solve this problem, the second component called "swap judge" is proposed, which is shown in Figure 4 (c). If $Addr_{choose} = Addr_{write}$, the page at $Addr_{choose}$ is written directly. Otherwise, a swap between page-A and page-B is triggered followed by a write to $Addr_{choose}$ ("swap-then-write"). In fact, the overhead of the "swap-then-write" can be reduced. We suppose the address of the unchosen page is $Addr_{not\_choose}$. "swap-then-write" can be done as follows: data at $Addr_{choose}$ is migrated to the page at $Addr_{not\_choose}$ followed by a write to $Addr_{choose}$. Thus, three writes introduced in original 'swap-then-write" are reduced to two writes.

Figure 4 does not specify the rule how writes are reallocated between toss-up pairs. To distribute the writes between pairs, we adopt a straightforward scheme called inter-pair swap: the page is swapped with a page at a random address every *"Inter-pair-swap-interval"* writes.

Toss-up can introduce enormous swaps. Theoretically, if the endurance of page-A or page-B and the write sequence are random, the possibility of swap is about $\frac{1}{2}$, which introduces a considerable write overhead [6, 13]. To mitigate the overhead, we should first analyze the factors affecting the swap frequency. In the next section, we establish a simple model to investigate factors influencing the possibility of the swap.

4.2 Theoretical Analysis of Swap Frequency

We can use a simple mathematical model to analyze the probability of the swap. As shown in Figure 4 (a), without loss of generosity, we suppose $E_A \geq E_B$. And we suppose that the probability of "$Addr_{write} = Addr_A$" is p, therefore the probability of "$Addr_{write} = Addr_B$" is $1-p$. Then, we calculate the chance of a swap in a single write as follows:

$$Prob(swap) = p \times \frac{E_B}{E_A + E_B} + (1-p) \times \frac{E_A}{E_A + E_B} \quad (1)$$

$$= \frac{p + (1-p) \times (E_A/E_B)}{1 + (E_A/E_B)} \quad (2)$$

According to Equation 2, there are four typical conditions regarding the values of p and $\frac{E_A}{E_B}$:

- Case-1: If $E_A \approx E_B$, $Prob(swap) \approx \frac{1}{2}$.
- Case-2: If $E_A \gg E_B$ and $p \to 1$, $Prob(swap) \approx 0$.
- Case-3: If $E_A \gg E_B$ and $p \to 0$, $Prob(swap) \approx 1$.
- Case-4: If $p \to \frac{1}{2}$, $Prob(swap) \approx \frac{1}{2}$.

Case-1 indicates that when E_A and E_B are close, the probability of swap is approximately $\frac{1}{2}$. Case-2 is the most preferred scenario with nearly no swap. "$p \to 1$" indicates the writes are consistent. After Case-3 occurs, page-A and page-B swap with each other and the situation turns into Case-2. For Case-4, if the $Addr_{write}$ keeps switching between $Addr_A$ and $Addr_B$, the frequency of swap is always close to $\frac{1}{2}$. This occurs when the write addresses are consecutive or random.

Based on the above analysis, we conclude as follows and propose two optimizations in the next section:

1. According to Case-1, Case-2 and Case-3, to reduce the swap frequency, two pages with distinct different endurance should be bond. Thus, we propose *Strong-Weak Paring* (SWP) to pair pages.
2. According to Case-4, to avoid that the swap frequency is always close to $\frac{1}{2}$, we can reduce the swap frequency by triggering the toss-up with an interval called *Toss-up Interval*. We call this technique Interval-triggered Toss-up.

4.3 Optimizations to Toss-up Wear Leveling

In this section, we specify Strong-Weak Paring (SWP) and Interval-triggered Toss-up, which are introduced to reduce the overhead of TWL's swap operations:

- **Strong-Weak Paring:** Pages are first sorted by their endurance. We suppose the sorted endurance table is $\{(PA_{i_1}, E_{i_1}), (PA_{i_2}, E_{i_2}), ..., (PA_{i_N}, E_{i_N})\}$. Then, pages at PA_{i_k} and $PA_{i_{N+1-k}}$ are bond with each other as a toss-up pair.
- **Interval-triggered Toss-up:** "Toss-up" in Figure 4 (b) is triggered every "Toss_up Interval" writes. Obviously, only when "Toss-up" functions, there is a possibility of $Addr_{choose} \neq Addr_{write}$ which triggers a swap (Figure 4 (c)). Therefore, the frequency of swap operations is reduced.

SWP can also improve the lifetime of PCM. If strong pages are coupled with weak ones, writes are dispatched to strong pages. Thus, weak pages are likely to suffer less writes. Effectiveness of above optimizations are evaluated in Section 5.

4.4 Overview of Read/Write Flow

In this section, we introduce the read flow and the write flow of PCM after TWL is adopted, which is shown in Figure 5. An entry of any tables in Figure 5 corresponds to a page. And LA/PA corresponds to logical/physical page address. In this work, we suppose that the granularity of writes is a memory page and data comparison write is employed [16].

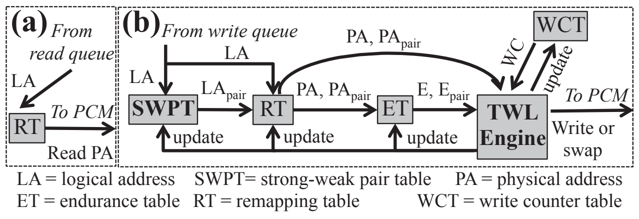

Figure 5: The overview of Toss-up Wear Leveling: (a) the flow of a read (b) the flow of a write

As shown in Figure 5 (a), the read flow is straightforward. Remapping table (RT) is queried with LA from the read queue. Then, PA is sent to PCM for corresponding data. The write flow is illustrated in Figure 5 (b). The LA to write is first sent to strong-weak pair table (SWPT). Then, LA's paired address LA_{pair} is found and forwarded to RT together with LA. After that, PA and PA_{pair} are sent to endurance table (ET) for corresponding endurance information E and E_{pair}. Then, TWL engine, which is shown in Figure 4, can perform swap or write operations to PCM regarding PA , PA_{pair}, E and E_{pair}. The write counter table (WCT) are also queried to determine if "toss-up" should be triggered or not according to the Toss_up Interval.

5. EVALUATION

In this section, experimental setup is first introduced. Then, we present the lifetime of PCM under various kinds of attacks to demonstrate the robustness of TWL, along with the vulnerability of prior works. We also illustrate the effects of our optimizations in swap overhead and lifetime. Last, we use PARSEC benchmark to evaluate the lifetime and performance overhead, followed by an analysis of TWL's hardware overhead.

5.1 Experiment Setup

Detailed parameters are listed in Table 1. A 8-core O3 CPU and a 32GB PCM are employed to represent a typical application scenario for servers [9]. For the PCM, we adopt parameters from [9, 7] as the state-of-the-art configurations. We assume that the endurance variation follows a Gauss distribution while endurance information is tested and stored at the granularity of page-size [1]. The mean endurance is 10^8 and the standard variation is 11% of the mean [6].

We adopt Bloom-filter based wear leveling (BWL) [13] and Security refresh (SR) [12] as the state-of-the-art works of PV-aware and traditional wear leveling schemes, respectively. In order to maximize the efficiency of BWL and SR, we also derive suggested parameters in [13] and [12]. For Toss-up wear leveling (TWL), we set the technology node at $32nm$ and derive the latency and logic gates with Synopsys [2]. To provide a

Table 1: Detailed simulation setup.

Processor Configuration
8-cores, ALPHA, Out-of-Order, 2GHz, Issue Width: 16, Fetch Width: 16, INT/FP FUs: 8/8, LD/ST: 24/24
Cache Configurations
DL1/IL1: 32/32KB, 2-way, 64B, R/W: 2/2-cycle, private L2: 2MB, 8-way, 128B, R/W: 10/10-cycle, share
PCM Configurations [7, 9]
32GB PCM, 4KB-page, 128 Byte per line, 4 ranks, 32 banks read/set/reset latency: 250/2000/250-cycle
TWL Configurations [12, 10]
Inter-pair swap interval: 128, RNG latency: 4-cycle TWL control logic latency/ table latency: 5/10-cycle

fair comparison, we fix the inter-pair swap interval (Section 4.1) at 128 [12]. We also list results of lifetime and performance when wear leveling is not adopted, which is labeled as "No wear leveling (NOWL)".

For benchmark evaluation, we choose PARSEC benchmark suite [3] which is shown in Table 2. We calculate the ideal lifetime, the time when all pages are worn out under corresponding write bandwidth. We first collect memory access traces from *gem*5 [4] by running each benchmark for one billion instructions. Then, we use the trace to simulate each benchmark's execution in loops until a PCM page wears out. We record the execution time as the lifetime of PCM. Table 2 lists the lifetime without wear leveling execution. For performance evaluation, we employ the full system mode of *gem*5 which is connected to NVMain [8].

Table 2: Benchmarks used in this work

Benchmark (PARSEC)	Write Bandwidth (MBps)	Ideal lifetime (years)	Lifetime w/o WL (years)
blackscholes	121	446	14.5
bodytrack	271	199	8.0
canneal	319	169	2.9
dedup	1529	35	2.5
facesim	1101	49	3.0
ferret	1025	52	1.2
fluidanimate	1092	49	2.0
freqmine	491	110	6.4
rtview	351	154	5.4
streamcluster	12	4229	132.2
swaptions	120	449	12.8
vips	3309	16	0.9
x264	538	100	2.0

5.2 Evaluation of Attacks

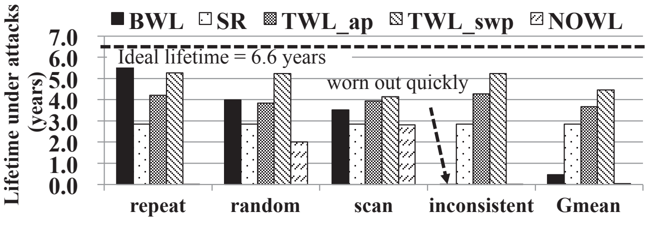

Figure 6: Lifetime under attacks using different wear leveling schemes

To comprehensive illustrate the robustness of TWL, we use four attack modes. The first three are from [11] and the last one is *inconsistent write* which is proposed in this work:

- **Repeat write mode :** Fix one address to write.
- **Random write mode:** Write addresses are random.
- **Scan write mode:** Write addresses are consecutive.
- **Inconsistent write mode:** The distribution of write intensity reverses before and after swaps (Section 3.2).

In Figure 6, we illustrate the years to wear-out the PCM under above attacks. Here we assume that the write stream is nonstop with an approximate 8GB/s write bandwidth, which indicates an ideal lifetime of 6.6 years. To demonstrate the improvement of strong-weak pairing (labeled as "TWL_swp"), we also evaluate the lifetime when adjacent (physical) pages are paired (labeled as "TWL_ap"), which is a naive scheme. In inconsistent write mode, PCM adopting BWL breaks down in 98 seconds since the prediction is always misled. SR achieves a lifetime of approximate 2.8 years under all attacks, which is mainly due to the wear-out of the weakest page. Compared to 'TWL_ap", a 21.7% lifetime improvement is achieved by "TWL_swp" on average . Among all the attacks, "TWL_swp" achieves minimum lifetime of 4.1 years under scan attack since enormous swaps are introduced (Section 4.2).

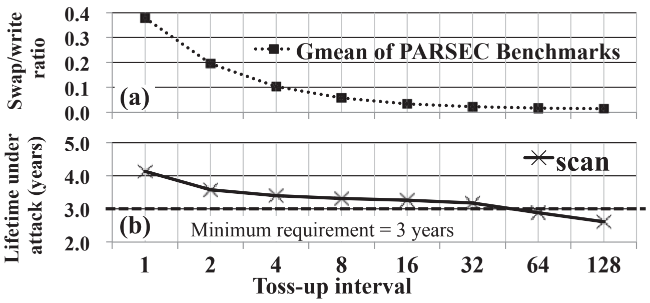

Figure 7: Metrics to choose the suitable toss-up interval: (a) swap/write ratio and (b) lifetime

We also evaluate the number of writes introduced by swaps when different toss-up intervals (Section 4.3) are employed. Figure 7 (a) shows the average ratio of swap writes (= the number of swaps) to the number of write requests using PARSEC benchmarks to make sense. For simplicity, in Figure 7 (b) we only list the lifetime under scan attack since lifetime under other attacks are higher and show a similar trend. When toss-up interval is one, the ratio can be as much as 37.9%, which introduces unacceptable write overhead. The ratio drops in proportion as the toss-up interval increases. However, as shown in Figure 7 (b), the lifetime of PCM also decreases. To meet the minimum requirement for servers' replacement cycles which is three to four years [12], we set the toss-up interval as 32 for the rest of evaluation, which incurs about 2.2% additional writes due to swaps.

5.3 Benchmark Evaluation

In Figure 8, lifetime are all normalized to the ideal lifetime in Table 2. We can find that security refresh achieves an approximate 44% lifetime of ideal. A manifest improvement in lifetime is demonstrated when Toss-up wear leveling or bloom-filter based wear leveling are adopted. On average, BWL achieves 75.6% of ideal lifetime while TWL achieves 79.6% .

We illustrate the performance overhead of wear lev-

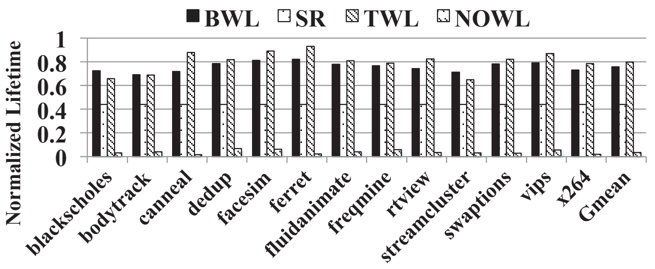

Figure 8: Normalized lifetime

eling schemes in Figure 9. All the execution time are normalized to that of "NOWL" scheme. For BWL, two bloom filters and a cold-hot list are accessed during every write [13]. By contrast, TWL engine functions only when write counter equals the toss-up interval, which introduces less write overhead. Timing overhead of TWL is at most 2.7% (*vips*). On average, TWL introduces 1.90% timing overhead while BWL and SR introduce 6.48% and 1.97% overhead, respectively.

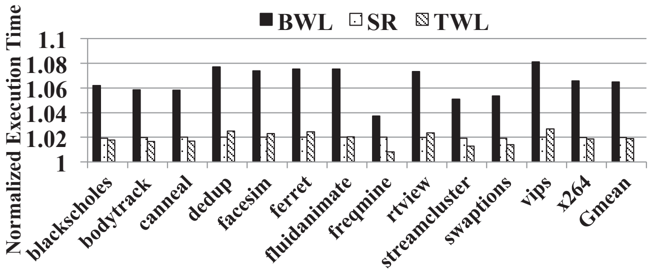

Figure 9: Normalized execution time with different countermeasures

5.4 Evaluation of Design Overhead

We also evaluate the design overhead of TWL in storage and logic gates. For TWL, as shown in Figure 5, a 7-bit write counter table entry, a 27-bit endurance table entry, a 23-bit remapping table entry and a 23-bit strong-weak pair table entry should be reserved for every PCM page. Thus, the storage overhead is about 80bits/4KB $= 2.5 \times 10^{-3}$.

TWL mainly consists of two logic parts. One is the random number generator and the rest includes a divider and several comparators. In our design, an 8-bit width Feistel Network is adopted to generate random numbers, which costs less than 128 gates [10]. The rest of logics costs 718 gates according to our synthesis results. Thus, 840 logic gates are estimated for the total logic gate cost of our design.

6. CONCLUSION

In this paper, we propose a novel wear-out attack towards previous PV-aware wear leveling, which explores the vulnerability of "prediction-swap-running" flow. We demonstrate that the vulnerability can be utilized by inconsistent write patterns to speedup the wear-out of weak cells. To mitigate the attack, we propose Toss-up Wear Leveling to impose writes according to the ratio of endurance within a bonding pair. Experiments demonstrate that Toss-up Wear Leveling can ensure PCMs' adequate lifetime under various attack, with negligible overhead in performance and hardware design.

7. REFERENCES

[1] M. Asadinia, M. Arjomand, and H. Sarbazi-Azad. Od3p: On-demand page paired pcm. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[2] H. Bhatnagar. *Advanced ASIC Chip Synthesis: Using Synopsys® Design CompilerTM Physical CompilerTM and PrimeTime®*. Springer Science & Business Media, 2007.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[5] M. Cintra and N. Linkewitsch. Characterizing the impact of process variation on write endurance enhancing techniques for non-volatile memory systems. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 217–228. ACM, 2013.

[6] J. Dong, L. Zhang, Y. Han, Y. Wang, and X. Li. Wear rate leveling: lifetime enhancement of pram with endurance variation. In *Proceedings of the 48th Design Automation Conference*, pages 972–977. ACM, 2011.

[7] F. Huang, D. Feng, W. Xia, W. Zhou, Y. Zhang, M. Fu, C. Jiang, and Y. Zhou. Security rbsg: Protecting phase change memory with security-level adjustable dynamic mapping. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, IEEE, 2016.

[8] M. Poremba and Y. Xie. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397. IEEE, 2012.

[9] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras. Preset: Improving performance of phase change memories by exploiting asymmetry in write times. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 380–391. IEEE, 2012.

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

[11] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini. Practical and secure pcm systems by online detection of malicious write streams. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 478–489. IEEE, 2011.

[12] N. H. Seong, D. H. Woo, and H.-H. S. Lee. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 383–394. ACM, 2010.

[13] J. Yun, S. Lee, and S. Yoo. Bloom filter-based dynamic wear leveling for phase-change ram. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1513–1518. EDA Consortium, 2012.

[14] W. Zhang and T. Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 2–13. ACM, 2009.

[15] M. Zhao, L. Jiang, Y. Zhang, and C. J. Xue. Slc-enabled wear leveling for mlc pcm considering process variation. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[16] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *In International Symposium on Computer Architecture*, 2009.