Lamina: Low Overhead Wear Leveling for NVM with Bounded Tail

Jiacheng Huang*, Min Peng*, Libing Wu†, Chun Jason Xue‡, Qingan Li*✉
*School of Computer Science, Wuhan University, China
†School of Cyber Science and Engineering, Wuhan University, China
‡Department of Computer Science, City University of Hong Kong, Hong Kong
{jiachenghuang, pengm, wu}@whu.edu.cn, jasonxue@cityu.edu.hk, qingan@whu.edu.cn

Abstract— Emerging non-volatile memory (NVM) has been considered as a promising candidate for the next generation memory architecture because of its excellent characteristics. However, the endurance of NVM is much lower than DRAM. Without additional wear management technology, its lifetime can be very short, which extremely limits the use of NVM. This paper observes that the tail wear with a very small percentage of extreme deviation significantly hurts the lifetime of NVM, which the existing methods do not effectively solve. We present Lamina to address the tail wear issue, in order to improve the lifetime of NVM. Lamina consists of two parts: bounded tail wear leveling (BTWL) and lightweight wear enhancement (LWE). BTWL is used to make the wear degree of all pages close to the average value and control the upper limit of tail wear. LWE improves the accuracy of BTWL by exploiting the locality to interpolate low-frequency sampling schemes in virtual memory space. Our experiments show that compared with the state-of-the-art methods, Lamina can significantly improve the lifetime of NVM with low overhead.

I. Introduction

Main memory technology plays a crucial role in computer architecture. Emerging non-volatile memory (NVM) technologies such as phase change, spin-torque transfer, resistive memories [7], [15] bring new opportunities to the main memory architecture design [17]. Because of the characteristics of high capacity, low energy consumption, and byte addressing ability, NVM becomes the competitor of traditional DRAM. However, the endurance of NVM cells is much lower than DRAM. For example, the write limit of PCM based NVM is between $10^7 - 10^8$ [12]. By contrast, the write endurance of DRAM is above 10^{15} [5]. Moreover, the write accesses to memory are commonly highly skewed. Fig. 1 shows the percentile of the number of memory write accesses over different pages within one minute. The selected benchmarks are from SPEC 2017 [2]. It illustrates that a very small percentage of the pages were written too much. The skewed write distribution on NVM memory will further reduce its lifetime, which greatly limits its application.

In order to prolong the lifetime of NVM, many wear leveling algorithms [19], [13], [5], [4], [9], [12], [10], [16], [3] have been proposed to make the write access evenly distributed on all cells of NVM. These work can be classified into two categories, age-based methods [9], [3], [19] and randomization-based ones [13], [12], [5], [10]. The age-based methods distinguish severely worn NVM areas from slightly worn ones, and try its best to put the new write accesses in the slightly worn areas. Segment swapping [19] records the write count of each segment. If a segment is written too many times,

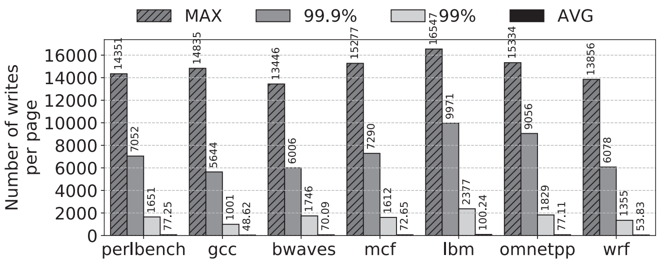


Fig. 1: The percentile of write count to each memory page. Taking *gcc* for example, the write count of the hottest page is 14835, while that of a nearly hottest page ranked at top 0.1% is 5644, and that of a page ranked at top 1% is 1001. It shows writes to memory pages is extremely unbalanced.

this segment is swapped with the least used segment with the help of an address mapping table. UWLalloc [9] records the allocation counts of memory blocks and tries its best to allocate memory blocks with the smallest allocation count upon memory request. The randomization-based methods try to put writes into the memory areas randomly. Security refresh [13] dynamically generates random keys to change the address mapping. Start-Gap [12] uses two registers (start register and gap register) to swap the selected memory area with its adjacent area. Kevlar [5] uses a random shuffle method to achieve the uniform distribution of write accesses.

We propose Lamina, an OS-level memory wear management scheme, which consists of two major modules. The first is bounded tail wear leveling, which is designed based on the idea of equal margin and tries to make the result of wear leveling approximate the ideal. The second is lightweight wear enhancement, which is used to improve the accuracy of low-frequency sampling schemes by exploiting the locality principle of virtual memory space to interpolating the sampling points. This module can improve the sampling accuracy, and thus improve the effect of wear leveling with low overhead.

We used SPEC 2017 benchmarks to evaluate the proposed work. Experimental results show Lamina can prolong the lifetime of NVM by $81.4\times$ on average with low performance overhead.

Our contributions can be summarized as follows:

- Based on the experimental evaluation, we observed that the lifetime of NVM is hurt by a small percentage of extreme tail wear, which existing methods does not effectively solve.
- We propose a novel bounded tail wear leveling method, which solves the problem of long tail wear.
- We propose a lightweight wear enhancement method,

which improve the sampling accuracy via interpolation, and finally improves the effectiveness of wear leveling with low overhead.

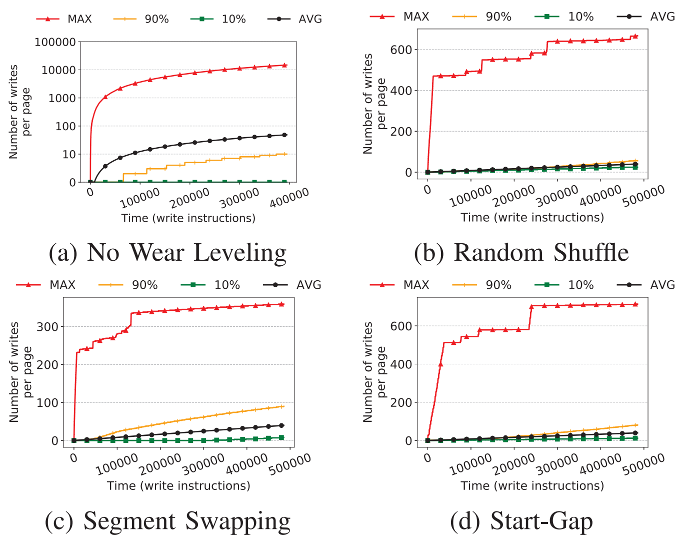
- Extensive experimental results show that Lamina can effectively prolong the lifetime of NVM with low overhead, compared to existing methods.

The rest of this paper is organized as follows. Section II describes observations and motivations. Section III details the design of Lamina. Section IV presents the evaluation results and analysis. Section V discusses the related work. We conclude this paper in Section VI.

II. Motivation

We co-design Lamina with memory management of operating system, which is mainly motivated by the following two observations.

A. *Tail Wear Problem*



(a) No Wear Leveling (b) Random Shuffle

(c) Segment Swapping (d) Start-Gap

Fig. 2: The percentile of the writes over time during program running. Although the existing methods can significantly reduce the write count of the hottest page, the hottest page still deviates far from the average write count, and the tail wear still exists.

Observation 1: Unbounded tail wear is the root cause that makes the actual wear leveling result deviate from the ideal result. When the system is running, the write distribution to memory is very uneven. As shown in Fig 2 (a), the average number of memory writes exceeds the 90% percentile. This result reflects that without wear leveling, a small part of the memory has been written frequently, while most of the other memory has been written very few times. Fig. 2 (b), (c), and (d) show the change of the percentile of write times over time using random shuffle, segment swapping, and start-gap wear leveling methods respectively. Although the maximum write times of memory pages decrease after using these three wear leveling methods, the results show that there is still a very large deviation between the maximum write number and the average write number (ideal). Furthermore, as the program runs, this deviation increases. The proposed work deals with the tail wear to improve wear leveling further.

B. *Dilemma between Performance and Accuracy*

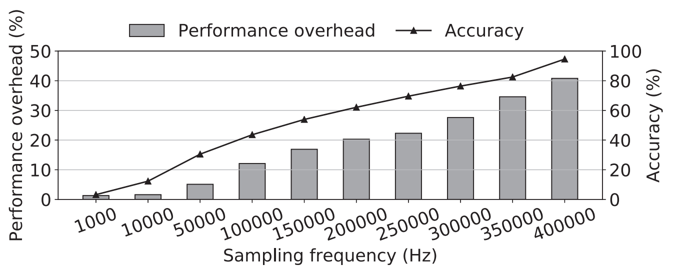


Fig. 3: Getting accurate write access information in real time will bring great performance overhead.

Observation 2: High frequency sampling will greatly degrade the performance of the system. Age-based methods generally require access information of memory pages. A very common method is to use a hardware based sampling method to get the memory access of programs during the system running [5]. However, sampling will bring unavoidable overhead to the system. To evaluate the overhead, a PEBS based method (detailed in Section III) is employed to sample the memory access of the system while running the *gcc* from the SPEC 2017 benchmark. As shown in Fig. 3, the performance overhead of the system increases as the sampling frequency increases. When the sampling frequency reaches 400kHz, the performance overhead approaches 40.7%. On the other hand, a low sampling frequency, commonly with low accuracy, may mislead the age-based wear leveling. Previous methods [5], [1] struggle to balance the trade-off between lower sampling cost and more accurate access information. This work proposes a lightweight wear enhancement method to improve the accuracy of low-frequency sampling schemes by interpolation, motivated by the locality principle in virtual memory space.

III. Design

A. *Lamina Overview*

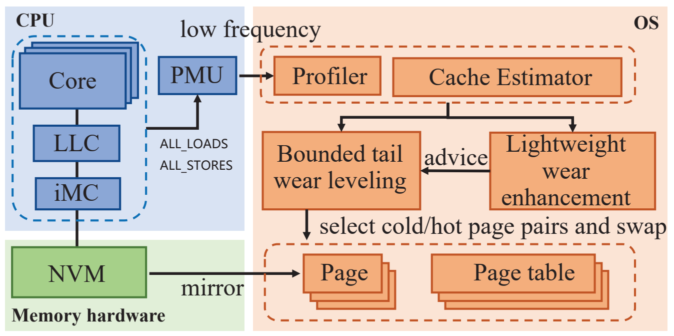


Fig. 4: An overview of Lamina.

The overview of the Lamina method is illustrated in Fig. 4. Lamina is implemented at the OS level and is mainly composed of two parts: bounded tail wear leveling (BTWL) and lightweight wear enhancement (LWE). In addition, Lamina also includes a profiler and a cache estimator, which are used for sampling memory operations and removing cache interference respectively (detailed below). These components

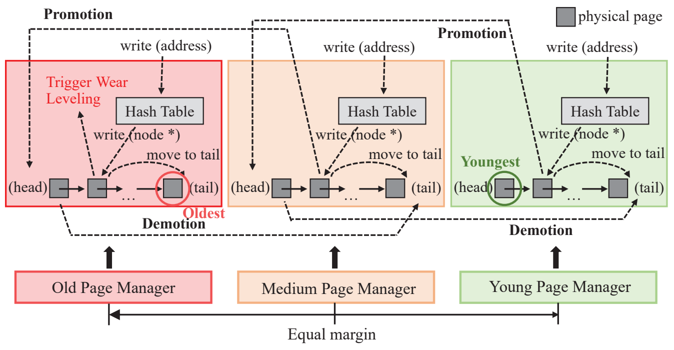


Fig. 5: Bounded tail wear leveling method.

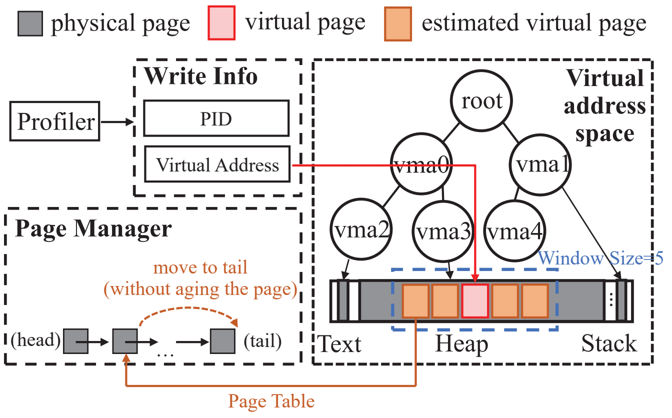


Fig. 6: Lightweight wear enhancement method.

cooperate with each other when the system is running. Profiler is used to interact with the PMU hardware for sampling memory instructions. The cache estimator is used to evaluate the actual write back of memory from LLC. BTWL uses the write access information obtained from sampling to discriminate the memory pages into different ages and manage them. LWE is responsible for enhancing the sampled data to estimate the write operations are lost during sampling.

The wear management process of Lamina is divided into three parts. First, we design a profiler to monitor the memory usage of all processes. So as to ensure the performance of the system, the profiler interacts with the PMU hardware at a low frequency. Because the write access to memory occurs when the data in the cache is written back, the cache estimator is used to evaluate the actual write back of memory. Second, BTWL uses the write access information obtained from sampling to discriminate the memory pages into different ages and manage them. When BTWL finds that the write distribution of memory pages is uneven to a certain extent, it triggers a wear leveling process, which selects appropriate pages to exchange the data and the indexes in the page table. Third, LWE also obtain the sampling information from the profiler. Its job is to enhance the sampled data. Many write operations are lost during sampling. LWE estimates the write operations that are not in the sampled data. The BTWL modifies the wear leveling strategy according to the estimation information generated from the LWE.

Profiler. This component collects the information of read and write operations by sampling. The collected data in the sampling includes PID, read/write type, virtual address, and physical address. Profiler mainly relies on two technologies related to Intel CPU hardware, namely PMU (Performance Monitoring Unit) and PEBS (Precise Event Based Sampling) [6]. A hardware PMC (Performance Monitoring Counter) is initialized on the PMU. PMC is automatically accumulated with the occurrence of specified hardware events. When PMC overflows, the PMU triggers a performance monitoring interrupt. With this interrupt, PEBS saves the current state of the CPU to memory. This enables us to collect the program execution information when the interrupt is triggered. In order to maintain the performance of the system, we set the frequency of this sampling as a small value in Lamina.

Cache Estimator. For the write address obtained by sampling, it is also necessary to distinguish whether the write operation is written to the cache or actually written to memory. Because the data retained in the cache only writes to memory once at write back, Lamina employed the method from Kevlar [5] to simulate the cache by using bloom filtering, and ensures that the probability of false positive is less than 1%.

B. Bounded Tail Wear Leveling

As shown in Fig. 5, BTWL divides the memory pages in the system into three generations: young, medium, and old. Each generation is managed by a page manager. Each page manager contains a linked list and a hash table. The linked list is responsible for managing the pages in the corresponding generation. Each node in the linked list contains a physical page frame and its age count. The hash table is used to handle the mapping of physical addresses to nodes. The page manager manage pages based on the position of pages in the linked list, which moves old pages towards the tail and young pages towards the head. The core idea is to dynamically promote and demote the pages among different generations when the system is running, so as to keep the age distances of the old and the young generation equal to the middle generation.

There are three key parameters: age, $base$, and $margin$. Each physical page has a corresponding age parameter saved in the corresponding linked list node, which represents the age of the physical page. The age parameter of each page is set to 0 during system initialization. The $base$ parameter represents the lower bound of the young page manager, which is also initialized to 0. The $margin$ parameter represents the age distance between the three page managers, and its value is initialized to a small arbitrary value (set to 10 in this article). In this way, the upper bound of tail wear is $base+2\times margin$.

The workflow of BTWL is mainly composed of three steps. First, BTWL periodically obtains the memory write access data sample set processed by the cache estimator from the profiler, which contains a series of write operations to memory pages during this period. For each sampled write address, BTWL finds the corresponding physical page through the hash table and increases the age count of the physical page by 1. Each page manager has a promotion threshold. The promotion thresholds of young, middle, and old page managers are $1/2/3\times margin+base$. If the age of the physical page is less than the promotion threshold, the page is inserted at the tail of the list. Conversely, the page is promoted and is inserted at the head of an older page manager.

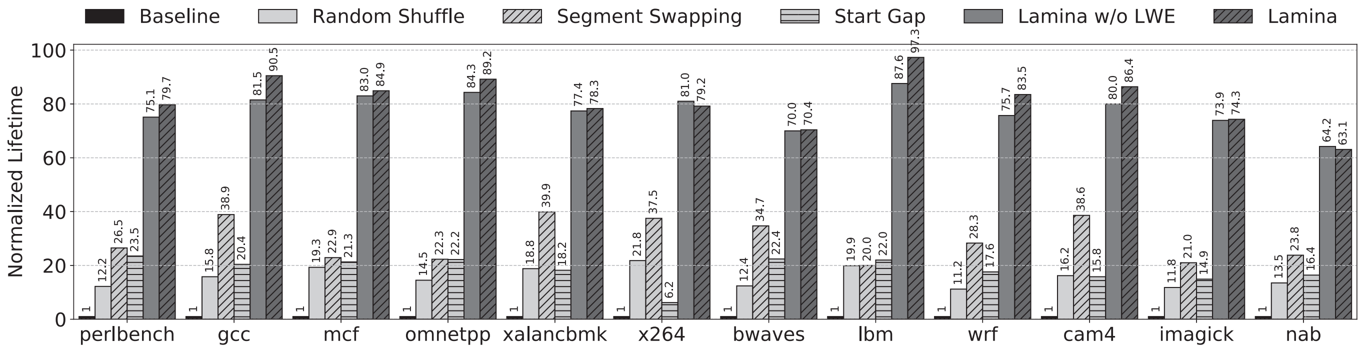


Fig. 7: The lifetime comparison of different wear leveling methods.

Second, if the promotion process is in the old page manager, it indicates that age of this page has reached the set bound. At this time, a page swapping process is triggered. It retrieves old pages and young pages from the tail of the old page manager and the head of the young page manager, respectively. For each pair of old and young pages, BTWL exchanges the data in the two pages through a temporary page on DRAM and then modifies the corresponding page table entries. For all pages involved in the swapping process, the page in the old page manager is inserted into the head, and the page in the young page manager is inserted into the tail. After the swapping process, the system set the base parameter value to the maximum age of the young pages participating in wear leveling. This work dynamically elevates the set bound according to the aging status of the whole memory.

Third, when the number of pages in the young page manager is less than that in the old page manager, the demotion process will be triggered. In this process, the old page manager and the medium page manager respectively select pages from the head and insert them into the tail of the younger page manager.

Through the above three steps, BTWL makes the write count of all pages tend to the average through three page managers with equal age margin and restricts the upper bound of memory page write times to $3 \times margin + base$. As a result, the wear leveling of the pages are expected to be close to the ideal at run time, and the lifetime of NVM can be improved.

C. Lightweight Wear Enhancement

As stated before, a low-frequency sampling scheme often incurs low accuracy, while a high-frequency sampling scheme often incurs high overhead. This work proposes a lightweight wear enhancement (LWE) method to improve the accuracy by exploiting the locality principle in virtual memory space, i.e., neighbour virtual pages share similar memory access behavior.

As shown in Fig. 6, the LWE method mainly consists of four steps. First, it obtains the PID and sampled virtual addresses corresponding to write operations from the low-frequency sampling based profiler. Second, it finds the corresponding virtual address space (mm_struct) and page table (PGD) of the process through PID. Third, according to the window size parameter, it finds the virtual pages neighbouring the sampled virtual page. Fourth, for each neighbor virtual page, it finds the corresponding physical page by looking up the page table and transferring this physical page to the tail of the list so as to reduce further reuse. The LWE chooses not to promote this page in purpose of tolerating the accuracy of interpolation.

Window size is a very important parameter, which will affect the accuracy of estimation and the efficiency of wear leveling. Too large window size will cause many memory pages without write access to participate in the wear leveling swapping operations, resulting in additional unnecessary page copies. On the other hand, if the window size is set too small, a lot of page write information would be lost, and many pages that have been written many times will be regarded as young pages mistakenly. In Section IV-D, we will analyze the influence of window size parameters on estimation accuracy through experiments, and explain how to select an appropriate window size value.

IV. Evaluation

A. Experimental Setup

In this section, we conducted extensive and representative experiments to evaluate the performance of Lamina. For experimental evaluation, our method and three previous representative methods, including Random Shuffle [5], [13], Segment Swapping [19], and Start-Gap [10], [12], are implemented at the OS level and evaluated.

The segment size of Segment Swapping is chosen as 10% of all written pages. The memory line size of Start-Gap is chosen as 64 pages (256KB). The Random Shuffle method shuffles the physical pages of all processes randomly. The period of these three wear leveling implementations are set to 10^4, 10^3, and 10^5 write instructions respectively.

These wear leveling methods are evaluated with the SPEC 2017 benchmark [2]. The test suites are run on a machine with an Intel Core i7-8750H processor and 32 GB DDR4 memory, with Linux kernel version 5.4.25. The DRAM memory in this machine is used as a simulation of NVM. The memory accesses from a very high frequency (1k$\times$ higher than Lamina) PEBS based sampling is used as the ground truth.

B. Lifetime Improvement

Fig. 7 shows the lifetime results of different wear leveling methods. We compare the maximum page write count of each wear leveling method with baseline (without wear leveling). Thus, we obtain the normalized lifetime results of these different wear leveling methods. According to this result,

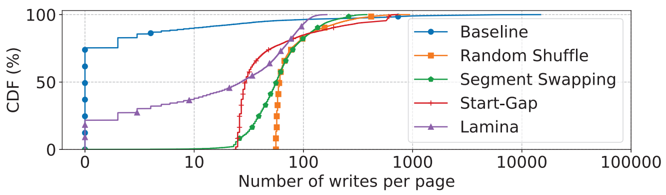


Fig. 8: The write distribution (using cumulative distribution function) comparison of different wear leveling methods.

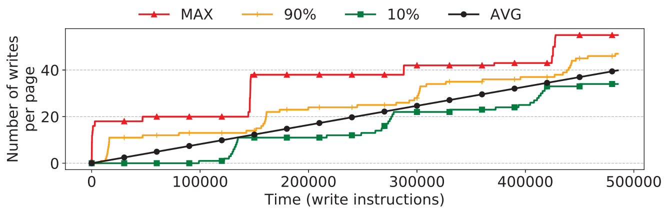


Fig. 9: The percentile of the write count of all pages as the program runs.

we can find that Lamina can improve the lifetime of NVM significantly compared with other methods. Compared to the baseline, Lamina can increase the lifetime of NVM up to 97.3 times for *lbm*, and 81.4 times on average. The experimental results also show that in most cases, using LWE to enhance the sampling data can improve the effect of Lamina. The exception is *nab*, for which the accuracy of the locality based interpolation slightly hurt the wear leveling.

C. Analysis of BTWL

Fig. 8 shows the distribution of memory page write times of *gcc* suite running for one minute under different wear leveling methods. From the result, we can see that in the case of Lamina, the number of tail writes of pages is less than that of other wear leveling strategies.

Lamina keeps approaching the average number of writes of all pages and limits the maximum number of writes. Since the additional overhead of wear leveling is mainly due to page swapping, we mainly consider the impact of additional memory writes and sampling process on system performance.

Fig. 9 shows the percentile of the memory page write count during one minute of *gcc* running when using Lamina for wear leveling. It is found that Lamina constantly approaches the average number of writes of all pages, and the maximum number of writes is well bounded. The results shown in Fig. 2 are the results of other wear leveling methods under the same conditions. Comparing these two figures, it is clearly shown that Lamina greatly eliminates the tail wear problem over other methods. It also explains why Lamina performs the best at improving the lifetime of NVM among all evaluated wear leveling methods.

D. Analysis of LWE

The LWE method can accurately interpolate the lost write information of sampling. Fig. 10 shows the accuracy of the write operations on the raw data obtained at a sampling frequency of 50000Hz, and the improved accuracy of LWE under different window size. Generally, the larger of the

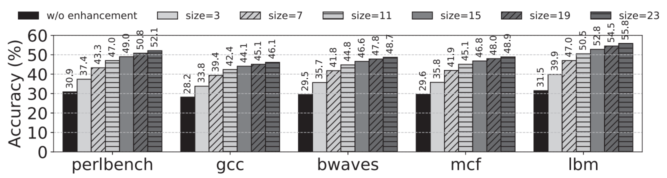


Fig. 10: The accuracy of LWE under different window size.

window size, the higher of the accuracy. If the window size is too large, the accuracy will converge. This also explains the increase of NVM lifetime after using LWE, as illustrated in Fig. 7. We generally set the window size to 7 in the experiment, because the accuracy of will converge over this window size.

E. Overhead Discussion

The performance overhead of wear leveling is mainly page swapping and sampling, so we mainly consider the impact of the additional memory writes due to these two factors. The performance overhead due to wear leveling is shown in Fig. 11. The experimental results show that compared with other wear leveling algorithms, Lamina incurs the least overhead. Since in Lamina the most appropriate memory pages are selected to achieve better wear leveling, and thus the number of page swapping is minimized. In addition, Lamina also uses a low-frequency sampling method which incurs less overhead. We can also observe that in all cases, the performance will be slightly reduced after using LWE. This is because LWE provides more memory write estimates, so more candidate pages will be selected when performing wear leveling.

V. Related Work

System design of Lamina has draw lesson from many previous wear leveling works [5], [4], [19], [9], [13], [12], [10], [16], [18], [3]. Some of these methods are age-based [9], [3], [19], which guides the wear leveling strategy through age counting. The method based on randomization [13], [12], [5], [10] is often easier to implement, which is used in a large number of system design. These methods can greatly prolong the service life of nonvolatile memory. However, with the running of the system, the imbalance degree of write distribution in NVM would constantly increase. There is a great gap between these wear leveling results and the optimal case. This paper bridged this gap to solve problems existing in the previous methods.

Analogy wear management methods have been proposed in the field of SSDs. For example, Rejuvenator [11] uses a fine-grained management strategy to divide blocks into different segments according to their age, so as to identify hot data and put them in blocks with less wear. Li et al. [8] proposes a data heat identification and aggregation migration method to balance the erasure times between SSD blocks. PER-WL [14] performs swapping between data blocks according to the statistical results of the program error rate to improve the efficiency of wear leveling. Unfortunately, these studies ignore many completely different characteristics of NVM in hardware and software levels, including access mode, system-related memory management, program performance requirement, etc.

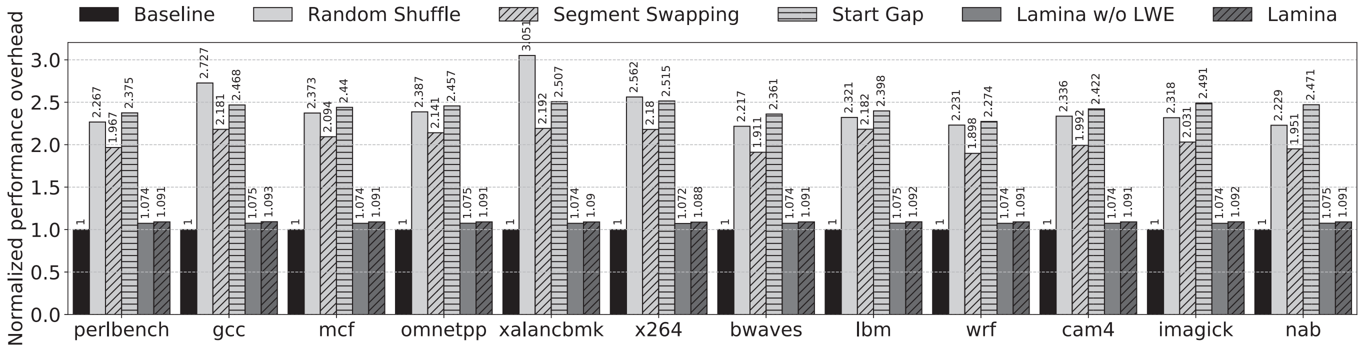


Fig. 11: The performance overhead of Lamina is very small. This is because it finds the most suitable pages for wear leveling and uses a low overhead sampling method. Compared with the comparative wear leveling methods, Lamina only introduces a small number of write overheads caused due to page swapping.

VI. Conclusion

In this paper, we present Lamina, which is an efficient operating system level memory wear management scheme. First, a bounded tail wear leveling algorithm is designed to make the average number of writes in each page close to the global average so as to solve the tail wear problem. Second, a lightweight wear enhancement method is proposed to estimate memory access information based on the locality principle in virtual memory, which can improve the sampling accuracy with low overhead. Our experimental results show that Lamina can greatly extend the lifetime of NVM at a very low cost.

Acknowledgments

This work was supported by CCF-Baidu Open Fund (NO.2021PP15002000), Key R&D Project of Hubei Province under Grant No. 2020BAA021, the National Natural Science Foundation of China (No. 62072346, 61972293, 91746206), and the Fundamental Research Funds for the Central Universities (2042020kf0217).

References

[1] Neha Agarwal and Thomas F Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 631–644, 2017.

[2] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018.

[3] Chi-Hao Chen, Pi-Cheng Hsiu, Tei-Wei Kuo, Chia-Lin Yang, and Cheng-Yuan Michael Wang. Age-based pcm wear leveling with nearly zero search cost. In *Proceedings of the 49th Annual Design Automation Conference*, pages 453–458, 2012.

[4] Xianzhang Chen, Zhuge Qingfeng, Qiang Sun, Edwin H-M Sha, Shouzhen Gu, Chaoshu Yang, and Chun Jason Xue. A wear-leveling-aware fine-grained allocator for non-volatile memory. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[5] Vaibhav Gogte, William Wang, Stephan Diestelhorst, Aasheesh Kolli, Peter M Chen, Satish Narayanasamy, and Thomas F Wenisch. Software wear management for persistent memories. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, pages 45–63, 2019.

[6] Intel. Microarchitecture Codename Nehalem Performance Monitoring Unit Programming Guide. https://software.intel.com/sites/default/files/m/5/2/c/f/1/30320-Nehalem-PMU-Programming-Guide-Core.pdf, 2020.

[7] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, 2009.

[8] Jun Li, Xiaofei Xu, Xiaoning Peng, and Jianwei Liao. Pattern-based write scheduling and read balance-oriented wear-leveling for solid state drivers. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 126–133. IEEE, 2019.

[9] Wei Li, Ziqi Shuai, Chun Jason Xue, Mengting Yuan, and Qingan Li. A wear leveling aware memory allocator for both stack and heap management in pcm-based main memory systems. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 228–233. IEEE, 2019.

[10] Haikun Liu, Yuanyuan Ye, Xiaofei Liao, Hai Jin, Yu Zhang, Wenbin Jiang, and Bingsheng He. Space-oblivious compression and wear leveling for non-volatile main memories. In *Proc. the 36th International Conference on Massive Storage Systems and Technology*, 2020.

[11] Muthukumar Murugan and David HC Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12. IEEE, 2011.

[12] 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 *2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 14–23. IEEE, 2009.

[13] 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. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 383–394, 2010.

[14] Xin Shi, Fei Wu, Shunzhuo Wang, Changsheng Xie, and Zhonghai Lu. Program error rate-based wear leveling for nand flash memory. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1241–1246. IEEE, 2018.

[15] Cong Xu, Dimin Niu, Xiaochun Zhu, Seung H Kang, Matt Nowak, and Yuan Xie. Device-architecture co-optimization of stt-ram based memory for low power embedded systems. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 463–470. IEEE, 2011.

[16] 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 the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[17] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 169–182, Santa Clara, CA, February 2020. USENIX Association.

[18] Xian Zhang and Guangyu Sun. Toss-up wear leveling: Protecting phase-change memories from inconsistent write patterns. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[19] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. *ACM SIGARCH computer architecture news*, 37(3):14–23, 2009.