Exploring Skyrmion Racetrack Memory for High Performance Full-Nonvolatile FTL

Ya-Hui Yang, Yu-Pei Liang*, Cheng-Hsiang Tseng, Shuo-Han Chen
Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan
*Institute of Information Science, Academia Sinica, Taipei, Taiwan
e-mail:romotty@gmail.com, *betty171920@iis.sinica.edu.tw, {t107590451, shchen}@ntut.edu.tw

***Abstract*—Skyrmion racetrack memory (SK-RM) has shown great potential for replacing DRAM or SRAM with its high density and great access performance. Unlike other non-volatile random access memory (NVRAM), SK-RM supports random updates through injecting or removing skyrmions on a racetrack. Injected skyrmions can be shifted along the racetrack to store data. Nevertheless, since most previous studies focus on minimizing the number of inject or shift operations, the unique feature of moving skyrmions vertically between racetracks and the possibility of utilizing free or invalid memory space to preserve skyrmions receive much less attention. In this paper, we observe that vertical movement and preservation of skyrmions provide a great opportunity to mitigate the possible run time performance degradation issue of NVRAM-based flash translation layer (FTL), as writes of mapping entries typically induce more overhead than reads on NVRAM. To fully exploit the benefits of SK-RM within the FTL mechanism, this paper proposes an SK-FTL to enable a high-performance and non-volatile FTL by preserving and reforming skyrmions over multiple data writes. Experimental results suggest that SK-FTL can effectively improve the performance of non-volatile FTL.**

***Index Terms*—SK-RM, skyrmions, racetrack memory, FTL, NAND Flash, NVRAM**

I. Introduction

Skyrmion racetrack memory (SK-RM) has been regarded as a great non-volatile random access memory (NVRAM) candidate in computer systems owing to its competitive read/write performance. Skyrmion is a topologically protected particle-like magnetization entry, which can be inserted/shifted/detected/removed on racetracks for storing data and supporting random information accesses and updates. As skyrmions can be shifted horizontally along racetracks or vertically between racetracks, SK-RM shows the possibility of preserving skyrmions in unused memory space and reforming those preserved skyrmions to compose new data pattern of future writes. Nevertheless, the methodology of preserving skyrmions in unused memory space over multiple data writes has not been explored before. Such observation motivates this paper to investigate and exploit the preservation and reformation of skyrmions to enhance the run time performance of the NVRAM-based flash translation layer (FTL). Meanwhile, since bit difference (i.e., the number of data bits 1) between two successive data writes leads to skyrmion insert and remove operations, the write data pattern to SK-RM should also be carefully tuned to avoid extra latency due to excessive skyrmion insert and remove operations. The technical difficulty of the proposed SK-FTL lies in *how to properly preserve and reuse skyrmions over multiple data writes and tune the bit pattern of FTL updates to avoid excessive skyrmion insert and remove operations.*

Flash translation layer (FTL) has been widely adopted by NAND flash-based storage devices, such as solid-state drives (SSDs), to manage the inherent constraints of NAND flash memory. The main task of FTL is to remap data write requests with logical block address (LBA) to allocatable NAND flash storage space with physical block address (PBA). Based on the emergence of NVRAM, NVRAM-based FTL mechanisms [3, 7, 9] have been proposed to host FTL on NVRAM, such as Phase Change Memory (PCM) [9] and Spin Torque Transfer (STT) RAM [8]. Nevertheless, according to Table I, writes on NVRAM usually induce longer latency than reads. In other words, when compared with DRAM-based FTL, *the lengthened write latency of NVRAM may degrade the run time performance of FTL.*

The emergence of SK-RM provides a great opportunity for resolving the run time performance degradation issue of NVRAM-based FTL, owing to the following reasons. First, as shown in Table I, the latency difference between read (detect) and write (insert) on SK-RM is much smaller than that of other NVRAM. Second, SK-RM can further lower the number of insert operations by allowing injected skyrmions to be reformed on the racetrack for composing the data pattern of the next data write and it is known as the permutation write [12]. Nevertheless, when the number of data bits 1 changes between successive data writes to SK-RM, the inject or remove operation is still required to either increase or decrease the number of skyrmions. This is because there is no extra region on racetracks for temporally caching skyrmions. To resolve such limitation without including additional racetracks, this paper proposes utilizing the free or invalid memory space on racetracks for preserving skyrmions. Unlike the permutation write, those skyrmions can be preserved over multiple data writes on the same racetrack as long as the memory space is unused. When the unused memory space is allocated, those preserved skyrmions can then be reformed to form the data pattern of the new data writes. Furthermore, this paper also proposes utilizing the vertical shift feature of SK-RM to preserve and reuse skyrmions between racetracks.

To benefit from the preservation and reformation capabilities of skrymions for non-volatile FTL, this paper proposes a SK-FTL to preserve skyrmions in unused memory space through both horizontal and vertical shift, while minimizing the bit difference between successive data writes through augmenting the space allocator of FTL. SK-FTL introduces three

TABLE I: Comparison of DRAM and NVRAM [3, 4, 8].

Latency	DRAM	PCM	STT-RAM	SK-RM
Read (ns)	15	50-70	1.62	0.1
Write (ns)	15	150-220	'0' to '1': 6 '0' to '1': 4	'0' to '1': 1.0 '1' to '0': 0.6

main components: (1) the SK-aware allocator, (2) the vertical shift mechanism, and (3) the active SK buffering scheme. The SK-aware allocator is first used to allocate free NAND flash space to incoming data writes while aiming to minimize the bit difference between the previously-stored PBA and the to-be-allocate PBA. Next, the vertical shift mechanism is included to manage the vertical movement of skyrmions between racetracks. Finally, the active SK buffering scheme is used for preserving injected skyrmions in unused memory space and reforming preserved skyrmions to compose the data pattern of new entry updates in FTL. Evaluation results show that SK-FTL can effectively reduce the accumulated insert and remove latencies of skyrmions by an average of 62.69% and 93.25%, when compared with native page-based FTL [1] on SK-RM with permutation write enabled [12].

The organization of this paper is summarized as follows. The background and motivation are described in Section II. The proposed SK-FTL is detailed in Section III. Next, experimental results are presented in Section IV. Finally, Section V concludes this paper with a few remarks.

II. Background and Motivation

A. NVRAM-based FTL

Owing to the high-performance and shock-resistance characteristics of NAND flash memory, NAND flash-based storage devices have gradually replaced hard disk drives (HDDs) as the mainstream storage medium. Different from HDDs, NAND flash memory has a few inherent limitations, which include the erase-before-write constraint, the limited number of program/erase (P/E) cycles, and asymmetric access/erase units. For instance, regarding the unit difference of NAND flash memory, the basic access unit is a *page*, while the minimum erase unit is a *block*, which consists of a few hundred pages. Meanwhile, due to the erase-before-write constraint, data updates cannot be written again to the same page and are typically written to other free pages through out-of-place updates. Such limitations require an additional management layer, namely the flash translation layer (FTL), within NAND flash-based storage devices for allowing NAND flash memory to be used as a block-based storage device.

One of the main components in FTL is the space allocator, which is utilized to comply with the erase-before-write constraint of NAND flash memory by redirecting data writes or updates to free pages. To track the physical pages of each data write, the mapping from logical block address to physical block address (LBA-to-PBA) is recorded by FTL as a mapping entry. Mapping entries are stored on dedicated NAND flash pages, which are also known as the metadata area, for persistence. On DRAM-based FTL, mapping entries are cached on the DRAM when the corresponding page is updated or accessed. Then, cached mapping entries on DRAM are written back to pages in the metadata area for persistence when entries are evicted or storage devices are powered off.

Loading and storing mapping entries not only induce additional internal traffics but also lead to the issue of write amplification. To alleviate the management overhead of mapping entries in FTL, nonvolatile random access memory (NVRAM) has been proposed to replace DRAM on NAND flash-based storage devices. Owing to the byte-addressability and nonvolatility of NVRAM, NVRAM can be used to store mapping entries and receive updates without rewriting other entries. As one of the main concerns of NVRAM is the lifetime, numerous NVRAM-based FTL mechanisms have been proposed with the consideration of NVRAM lifetime or bit-level wear leveling. For instance, Kim et al. [7] include PRAM in NAND Flash storage devices for establishing a high-performance embedded storage subsystem. Meanwhile, Liu et al. [9] propose a PCM-FTL to store the mapping table in the PCM instead of DRAM and focus on enhancing the endurance of the PCM by minimizing the number of bit flips in the PCM. More recently, Cheng et al. [3] adopt NAND-SPIN as the medium for NVRAM-based FTL and propose a window-based wear leveling to resolve the uneven bit-level wearing issue of NVRAM-based FTL. Nevertheless, since writes on NVRAM are typically more time-consuming than reads, *the possible run-time performance degradation on NVRAM-based FTL should also be examined thoroughly.*

B. Skyrmion Racetrack Memory

Skyrmion racetrack memory (SK-RM) has competitive advantages, including the non-volatility, similar access performance, and higher storage density when compared with SRAM or DRAM. SK-RM is an evolved version of domain-wall racetrack memory (DW-RM) and outperforms DW-RM in terms of cell size and current density [6] by utilizing topologically protected magnetic skyrmions for storing information. Data bits 1 and 0 are represented by the presence or absence of skyrmions on the racetrack, which has multiple bit zones for holding skyrmions. To support random updates on SK-RM, skyrmions are manipulated through current applied to the ends of the racetrack and access ports. The structure and operations of SK-RM are shown in Figure 1.

Figure 1(a) illustrates an example of SK-RM with 3 access ports and 11 bit zones. The presence and absence of skyrmions in bit zones are denoted as *skyrmion* and *non-skyrmion* for data bits 1 and 0, respectively. The distance between access ports on the racetrack is referred to as interport distance and is defined by the physical layout of the SK-RM. The regions located beside the right-most and left-most access ports are the overhead regions for temporally holding skyrmions out of the racetracks during manipulating skyrmions. Figure 1(b) then shows the details of shift operation. The shift current can be provided at both ends of the racetrack and access ports to either end of the racetrack and access ports to drive all the skyrmions and non-skyrmions along the direction of the applied current. Notably, it is also possible to shift skyrmions onto different racetracks through access ports [5]. Next, as shown in Figure 1(c), to insert skyrmions for representing data bits 1, skyrmions are

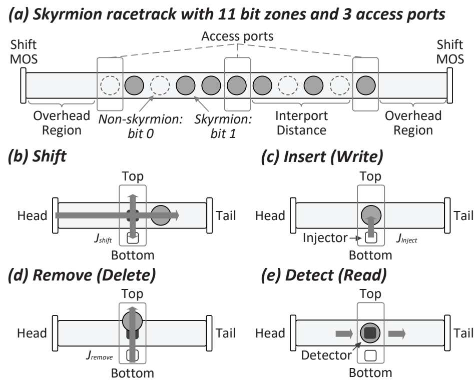

Fig. 1: **Skyrmion racetrack memory (SK-RM)**, *which is composed of racetracks and access ports. SK-RM supports random information accesses and updates through detecting, injecting, shifting and removing skyrmions.*

injected by the injector at the access port and shifted onto the racetrack. Oppositely, for data bits 0, no inject operation is needed as data bits 0 are represented by the absence of skyrmions. On the other hand, as shown in Figure 1(d), during the remove operation, removal currents are applied to either end of the racetrack or access ports to over-shift skyrmions out of other ends. For reading data bits on SK-RM, both skyrmions and non-skyrmions are shifted across the detectors at the intersection of access ports and the racetrack for reading the stored bits. The latency of each skyrmion operation can be summarized in Table II. Notably, the remove and insert latencies include one shift operation for shifting skyrmions onto or from race tracks.

TABLE II: Latency of SK-RM operations [4].

Operations	Read	Shift	Remove	Insert
Latency	0.1 ns	0.5 ns	0.8 ns	1 ns

As insert operation induces the longest latency among skyrmion operations, studies have been proposed to reduce the number of shift operations by optimizing the data layout strategies. For instance, Yang et al. [12] propose a new data update strategy, namely permutation write, for SK-RM through reforming the existing skyrmions to reduce the number of skyrmion injections for composing the next data pattern. On the other hand, Hsieh et al. [4] propose a back-to-back mechanism to minimize the shift operations during the sorting process. Even though strategies for reducing the number of shift and inject operations have been proposed, *this paper argues that the latencies of remove and inject operations should be considered at the same time between data writes because data bit difference between two successive writes could lead to excessive inject or remove operations over time.*

C. Motivation

Even though directly deploying FTL on SK-RM can take advantage of the short read/write latency of SK-RM, the bit pattern difference between successive FTL entry updates could lead to excessive insert and remove operations. Over time, the latency induced by these insert and remove operations could degrade the performance of SK-RM-based FTL. To better illustrate such condition, a motivational example is given in Figure 2. In Figure 2(a), the data chunk C is updated twice on the NAND flash memory. Based on the conventional page allocation policy, pages of each block are allocated sequentially for accommodating new data writes. Therefore, the physical location of data chunk C changes from [block 0, page 2] to [block 0, page 255], and then changes to [block 8, page 2]. Accordingly, as shown in Figure 2(b), the mapping entry of data chunk C is also updated. Owing to the 1-bit difference between entry updates, 7 skyrmions are injected during the first update. However, those injected skyrmions are removed immediately during the next update. In other words, if the number of data bits 1 between entry updates is different, insert and remove operations are induced. Excessive insert and remove operations may eventually aggravate the performance of SK-RM-based FTL. In summary, *the major issue of exploiting SK-RM for a high-performance nonvolatile FTL lies in how to preserve and reuse injected skyrmions properly while avoiding additional shift operations.*

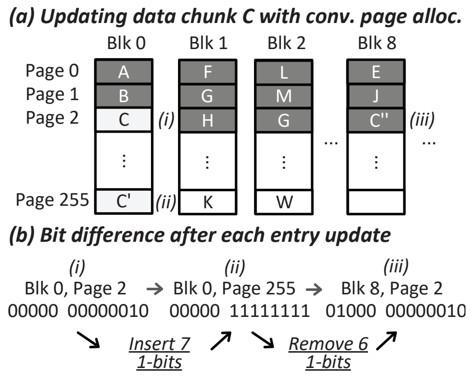

Fig. 2: **Motivational example**, *which shows the skyrmion insert and remove operations when the number of 1-bit varies between successive FTL mapping updates.*

III. SK-RM-based Flash Translation Layer

A. Overview

To exploit the preservation and reformation characteristics of skyrmions for enabling high-performance SK-RM-based FTL, this paper proposes an SK-FTL for minimizing the bit difference between successive mapping entry updates and preserving injected skyrmions to compose data pattern of future data writes. The main difference between the proposed SK-FTL and previous SK-RM data write strategy is that (1) SK-FTL first proposes a systematic methodology to exploit both preservation and reformation characteristics, and (2) SK-FTL aims to minimize the number of insert and remove operations simultaneously between mapping entry updates for further latency reduction. The system architecture of the proposed SK-FTL is illustrated in Figure 3.

Notably, SK-FTL is designed based on page-based FTL, and each mapping entry fits into the distance between access ports on the racetracks. Multiple racetracks are included, and the vertical shift of skyrmions can only be applied between two tracks. In other words, shifting skyrmions from racetrack

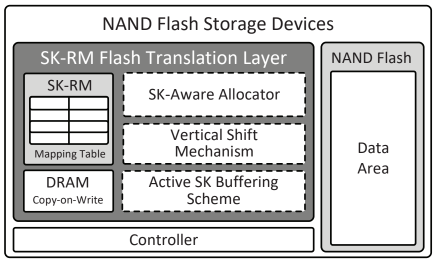

Fig. 3: **Structure of the proposed SK-FTL**, *in which SK-RM is used to store mapping entries of FTL and 3 main components are included to exploit the preservation and formation characteristics of SK-RM.*

N to racetrack N+2, two shift operations are required. Meanwhile, as the bit pattern between entry updates affects the number of insert and remove operations, SK-FTL minimizes the number of '1' bits difference by the SK-aware allocator (see Section III-B) to consider the 1-bit difference between the originally-stored PBA and to-be-allocated PBA. Then, the vertical shift mechanism (see Section III-C) is included to manage vertical shift between racetracks. In the end, the active SK buffering scheme (see Section III-D) preserves and reforms injected skyrmions for prolonging the lifetime of injected skyrmions.

B. *SK-Aware Space Allocator*

The proposed SK-FTL updates mapping entries by following the permutation write strategy. As shown in Figure 4, during each update, the to-be-updated mapping entry is first shifted to the overhead region on the racetrack. During the shifting, skyrmions, which represent data bits 1, are preserved in the overhead region, while non-skyrmions are removed. Then, skyrmions are shifted back to compose the data pattern of the next mapping entry update. In this example, there is no remove or insert operations because the number of 1-bit is identical between two successive updates. Nevertheless, according to Figure 2, the bit difference is common during mapping updates.

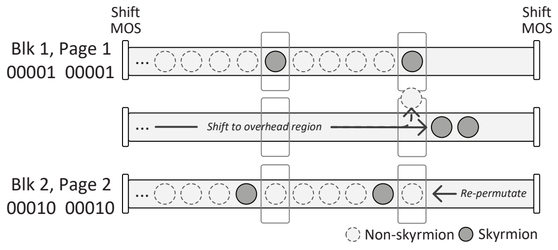

Fig. 4: **Mapping entry updates through permutation write**, *in which the data pattern of next mapping entry is recomposed based on injected skyrmions of previous entry.*

To minimize the bit difference between each mapping entry update, the included SK-aware space allocator groups NAND flash blocks into different lists based on their number of 1 bits in their block addresses. These lists are referred to as bits-number-based block lists. As shown in Figure 5, block numbers are converted into binary values at Step (a). This is because the binary value is the final form of how mapping entries are stored on SK-RM. Then, based on their number of 1 bits, blocks are added to each list. For instance, since Block 3 and Block 5 have 2 1 bits in their binary values of block addresses, Block 3 and Block 5 are added to list of 2 bits. Then, allocating free NAND flash pages from blocks on the same list will not change the number of 1 bits for the block address in mapping entries. In other words, as shown in Figure 4, through re-permuting existing skyrmions, it will not induce any insert or remove operations for updating the block address in mapping entries as there is no bit difference.

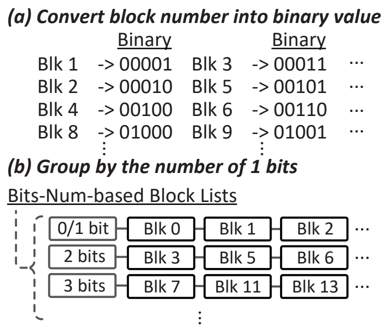

Fig. 5: **Bits-number-based block lists**, *which groups blocks that have same number of 1 bits in their block address together.*

After minimizing the possibility of 1-bit difference in block address of mapping entries, the SK-aware space allocator is proposed to allocate free NAND flash block horizontally across blocks in the same bits-number-based block list. By the horizontal allocation fashion shown in Figure 6, the bits difference between successive entry updates can be further reduced. For instance, since horizontal allocation ensures the page number is identical, there are no insert or deletion operations required while recomposing the data pattern of the page number in the next mapping entry update.

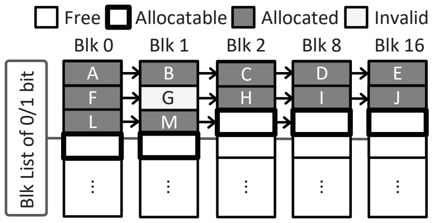

Fig. 6: **SK-aware space allocator**, *which allocates NAND flash pages horizontally across blocks to minimize the 1-bit difference of page address between mapping entry updates.*

C. *Vertical Shift Mechanism*

On SK-RM, vertically shifting skyrmions onto different racetracks through access ports has been successfully demonstrated [5]. However, few previous studies have utilized this feature for actual use because arbitrarily moving skyrmions to other racetracks may corrupt the original valid data on the racetrack. In other words, the main challenge for exploiting the feature of shifting skyrmions between racetracks is when and where the skyrmions can move without ruin the original data stored on the racetrack. The proposed SK-FTL resolves the above issue by utilizing the valid and invalid information of mapping entries within FTL as a hit to determine the timing and regions for shifting skyrmions vertically. The vertical shifting mechanism can be summarized in Figure 7.

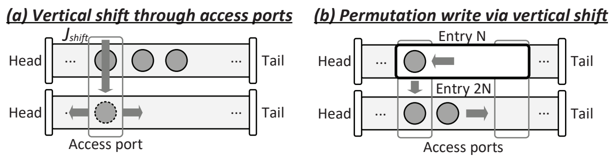

Fig. 7: **Vertical Shift Mechanism**, *which exploit and manage shifting skyrmions to other racetrack for either preservation or data pattern reformation.*

As shown in Figure 7(a), the vertical shift is an operation that allows shifting the skyrmions to adjacent racetracks through the access ports. Then, based on the vertical shift operation, the included vertical shifting mechanism repurposes other free or invalid memory regions on adjacent racetracks as the buffer region of permutation write strategy. For example, in Figure 7(b), Entry 2N is an unused entry in the mapping table. Then, when updating the Entry N, SK-FTL uses the space of Entry 2N as the buffer region to perform permutation write. In addition, since the memory space used as buffer region is either free or invalid, excessive skyrmions between mapping entry updates can be preserved. Alternatively, when skyrmion injections are required during entry updates, those preserved skyrmions in the buffer region can be reused.

D. Active SK Buffering Scheme

With the perpendicular shifting mechanism, SK-FTL can shift skyrmions vertically to invalid or free regions on adjacent racetracks and repurposes those regions are buffer regions while reforming the data pattern of next entry updates. More specifically, the SK-FTL repurposes the free and invalid entry in the mapping table as the buffer for performing permutation write and further preserves the extra skyrmions to avoid repeatedly removing and inserting the skyrmions. To better manage those to-be-preserved skyrmions, SK-FTL includes the active SK buffering scheme for preserving and recomposing skyrmions for future data writes. Figure 8 shows a working example of the active SK buffering scheme.

As shown in Figure 8, when a mapping entry is being updated, SK-FTL can use the memory space of the free and invalid entries on the same racetrack or the adjacent racetracks through the horizontal or vertical shift operations. For example, in Figure 8, when Entry 0 is going to be updated,

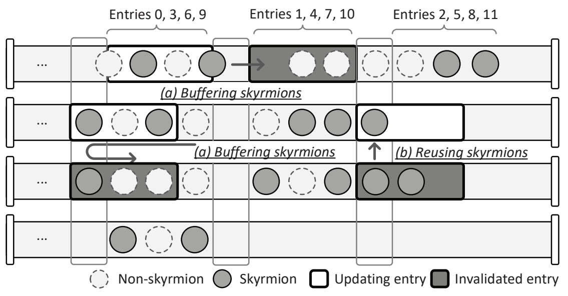

Fig. 8: **Active SK Buffering Scheme**, *in which skyrmions are preserved or reformed for future data writes.*

Entry 1 is chosen as the buffer region for temporally holding the skyrmions and re-forming the data pattern of updated mapping entries through the shift operations because Entry 1 is an invalid entry in the mapping table. Similarly, Entry 6 is used as the buffer region for the to-be-updated Entry 3 through the vertical shift operation. On the other hand, since skyrmions on the memory region of free or invalid entries do not represent actual data, those previously-injected skyrmions can be reused by other entries. For instance, in Figure 8, Entry 8 is an invalid entry and contains 3 previous-injected skyrmions. When Entry 5 needs skyrmion injections during entry updates, the skyrmions of Entry 8 can be reused to form the data pattern of updated Entry 5 without injecting new skyrmions.

IV. Performance Evaluation

A. Experimental Setup

This section demonstrates the evaluation of the proposed SK-FTL. To study the capabilities of SK-FTL, the evaluations are conducted by the realistic traces collected from Microsoft Research Cambridge (MSR) [10]. In addition, a self-collected trace that collected one-month I/O behavior of a personal computer is also included. All the comparisons are implemented in a flash simulator [2], including the FTL on SK-RM with the naïve write strategy, FTL on SK-RM with the permutation-write strategy [12], and our proposed SK-FTL. Above comparisons are denoted as *FTL*, *PW FTL*, and *SK-FTL*. Please note that the naïve write strategy of SK-RM refers to remove all previous-injected skyrmions and inject new skyrmions based on the updated data pattern. The size of the simulated flash is 64 GB with 16KB pages [11], and the latency parameters of SK-RM are shown in Table II.

B. Experimental Results

To evaluate the performance of SK-FTL, our experiments first measure the latency of four basic SK-RM operations during updating mapping entries of FTL, and the results are shown in Figures 9-12. In these figures, the x-axis represents different traces, and the y-axis is the accumulated latency of each operation in log scale with the nano-second unit while running different traces. The results show that SK-FTL can effectively reduce the skyrmion insert and remove latencies when compare to the other two methods. More specifically, when compared to PW FTL, SK-FTL can reduce the insert latency by 62.7% on average. In addition, regarding the remove latency, SK-FTL can reduce 93.3% on average when compared to PW FTL. The result of detect and shift operations are shown in Figure 11 and Figure 12, respectively. Since the native SK-RM write strategy always removing all the skyrmions and then re-inserting the skyrmions according to the new data, there is no need to detect any bits while adopting the naïve write strategy. Therefore, in Figure 11, the detect latency of FTL is always zero in every trace. On the other hand, as the result of the same detect behavior in both PW FTL and SK-FTL, the detect latency of these two methods are the same in the result. Last, in Figure 12, the shift latency of the three compared methods are similar because the number of shift operations depends on the interport distance.

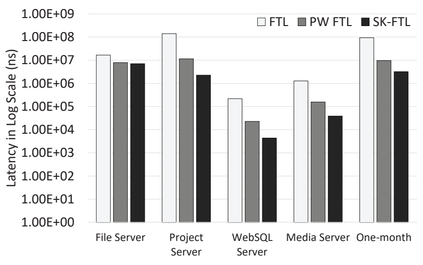

Fig. 9: Inject latency comparison.

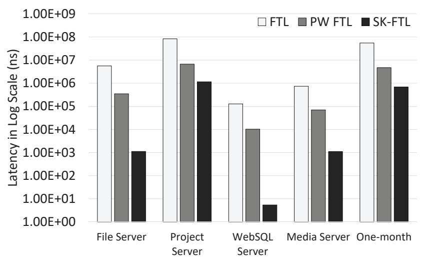

Fig. 10: Remove latency comparison.

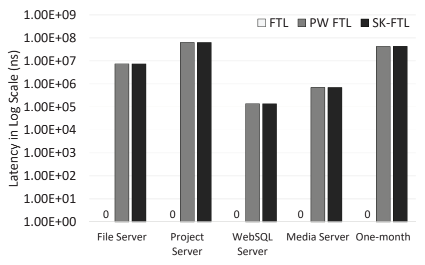

Fig. 11: Detect latency comparison.

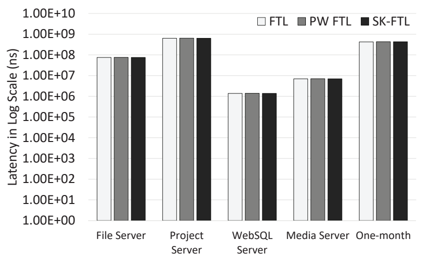

Fig. 12: Shift latency comparison.

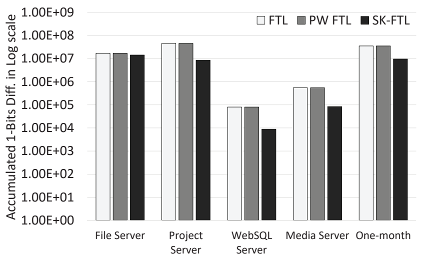

Fig. 13: Accumulated 1-bit difference.

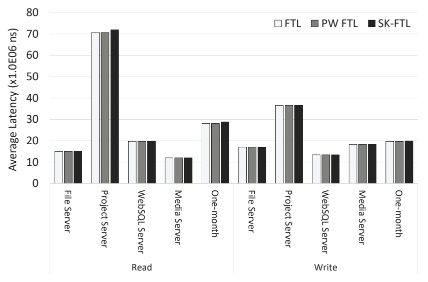

Fig. 14: Read/Write Latency of NAND flash.

Meanwhile, to demonstrate the benefit brought by the SK-aware allocation, the total number of 1 bits difference is also measured. As shown in Figure 13, SK-FTL can reduce the difference of 1 bits by 68.7%, compared with the FTL and PW FTL. On the other hand, the number of 1 bits differences of FTL and PW FTL are identical for each trace because their NAND flash page allocation strategies are not tuned for minimizing 1-bit difference. Furthermore, as FTL is an auxiliary layer for managing the NAND flash memory, the access performance of NAND flash memory is an important consideration. Therefore, the performance of NAND flash is also measured in terms of the average read and write latencies. As shown in Figure 14, SK-FTL can achieve similar performance on NAND flash memory as conventional FTL, and the differences are only 0.89% and 0.28% on average for read and write latencies, which are negligible when compared with the reduction of insert and remove latencies of SK-RM. Notably, the performance shows in Figure 14 only includes the latency of regular I/O operations on NAND flash, and the latency of FTL-related operations is not included.

V. Conclusion

To exploit the skyrmion preservation and reformation characteristics of SK-RM, this paper is a pioneer study that systematically utilizes the characteristics of SK-RM in combination with the mapping information of FTL for achieving high-performance nonvolatile FTL. The proposed SK-FTL first introduces the SK-aware space allocator to carefully minimize the 1-bit difference between mapping updates by configuring the allocation method of the NAND flash pages. Then, the vertical shift mechanism is used to repurpose invalid or free entry space as the buffer region for reforming skyrmions for the next entry update. Finally, the active SK buffering scheme is utilized to preserve skyrmions in unused space and recompose skyrmions for future data writes. Evaluation results show that the proposed SK-FTL can reduce the latencies of skyrmion insert and remove operations by 62.69% and 93.25% on average and induce no extra shift operation, compared with the page-based FTL on SK-RM with permutation write strategy enabled.

Acknowledgments

This work was supported by the Ministry of Science and Technology, Taiwan under grant no. 109-2222-E-027-007-MY3.

References

[1] A. Ban. Flash file system, U.S. 5404485 A, Apr. 1995.

[2] Y.-H. Chang and T.-W. Kuo. A management strategy for the reliability and performance improvement of mlc-based flash-memory storage systems. *IEEE Transactions on Computers*, Mar 2011.

[3] W.-C. Cheng, S.-H. Chen, Y.-H. Chang, K.-H. Chen, J.-J. Chen, T.-Y. Chen, M.-C. Yang, and W.-K. Shih. Ns-ftl: Alleviating the uneven bit-level wearing of nvram-based ftl via nand-spin. In *2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6, 2020.

[4] Y. S. Hsieh, P. C. Huang, P. X. Chen, Y. H. Chang, W. Kang, M. C. Yang, and W. K. Shih. Shift-limited sort: Optimizing sorting performance on skyrmion memory-based systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4115–4128, 2020.

[5] W. Kang, X. Chen, D. Zhu, X. Zhang, Y. Zhou, K. Qiu, Y. Zhang, and W. Zhao. A comparative study on racetrack memories: Domain wall vs. skyrmion. In *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 7–12, 2018.

[6] W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao. Skyrmion-electronics: An overview and outlook. *Proceedings of the IEEE*, 104(10):2040–2061, 2016.

[7] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng. A pram and nand flash hybrid architecture for high-performance embedded storage subsystems. In *Proceedings of the 8th ACM International Conference on Embedded Software*, 2008.

[8] Y.-P. Liang, T.-Y. Chen, Y.-H. Chang, S.-H. Chen, P.-Y. Chen, and W.-K. Shih. Rethinking last-level-cache write-back strategy for mlc stt-ram main memory with asymmetric write energy. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[9] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao. Pcm-ftl: A write-activity-aware nand flash memory management scheme for pcm-based embedded systems. In *IEEE 32nd Real-Time Systems Symposium*, 2011.

[10] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *ACM Transactions on Storage (TOS)*, 2008.

[11] Samsung. Samsung v-nand@ONLINE, http://www.samsung.com/semiconductor/products/flash-storage/v-nand/, 2017.

[12] T. Y. Yang, M. C. Yang, J. Li, and W. Kang. Permutation-write: Optimizing write performance and energy for skyrmion racetrack memory. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.