Adaptive Wear-Leveling Algorithm for PRAM Main Memory with a DRAM Buffer

SUNG KYU PARK and MIN KYU MAENG, KAIST
KI-WOONG PARK, Daejeon University
KYU HO PARK, KAIST

Phase Change RAM (PRAM) is a candidate to replace DRAM main memory due to its low idle power consumption and high scalability. However, its latency and endurance have generated problems in fulfilling its main memory role. The latency can be treated with a DRAM buffer, but the endurance problem remains, with three critical points that need to be improved despite the use of, existing wear-leveling algorithms. First, existing DRAM buffering schemes do not consider write count distribution. Second, swapping and shifting operations are performed statically. Finally, swapping and shifting operations are loosely coupled with a DRAM buffer. As a remedy to these drawbacks, we propose an adaptive wear-leveling algorithm that consists of three novel schemes for PRAM main memory with a DRAM buffer. The PRAM-aware DRAM buffering scheme reduces the write count and prevents skewed writing by considering the write count and clean data based on the least recently used (LRU) scheme. The adaptive multiple swapping and shifting scheme makes the write count even with the dynamic operation timing, the number of swapping pages being based on the workload pattern. Our DRAM buffer-aware swapping and shifting scheme reduces overhead by curbing additional swapping and shifting operations, thus reducing unnecessary write operations. To evaluate the wear-leveling effect, we have implemented a PIN-based wear-leveling simulator. The evaluation confirms that the PRAM lifetime increases from 0.68 years with the previous wear-leveling algorithm to 5.32 years with the adaptive wear-leveling algorithm.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles

General Terms: Design, Management, Experimentation

Additional Key Words and Phrases: Adaptive wear-leveling, PRAM main memory, DRAM buffering, swapping and shifting, PIN-based simulator

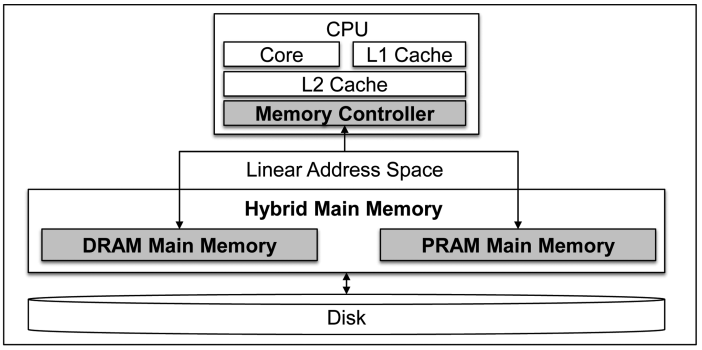
ACM Reference Format:
Sung Kyu Park, Min Kyu Maeng, Ki-Woong Park, and Kyu Ho Park. 2014. Adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer. ACM Trans. Embedd. Comput. Syst. 13, 4, Article 88 (February 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/2558427

1. INTRODUCTION

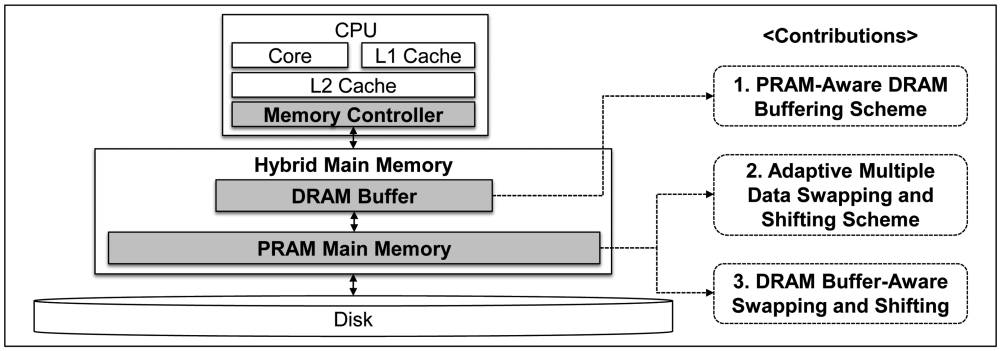
For several decades, DRAM has been the primary main memory used for applications ranging from mobile devices to large-scale computing systems. With the growth of new

A preliminary version of this work was published in *Proceedings of the 2012 ACM Symposium on Applied Computing (SAC'12)*.
The work presented in this article was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.
Authors' addresses: S. K. Park, M. K. Maeng, and K. H. Park, Department of Electrical Engineering, KAIST, 335 Gwahak-ro (373-1, Guseong-dong), Yuseong-gu, Daejeon, 305-701, Republic of Korea; email: {skpark, mkmaeng, kpark}@core.kaist.ac.kr; K.-W. Park, Department of Computer Hacking and Information Security, Daejeon University, 62 Daehak-ro Dong-gu, Daejeon, 200-716, Republic of Korea; email: woongbak@dju.ac.kr.

DOI: http://dx.doi.org/10.1145/2558427



(a) (DRAM + PRAM) main memory



(b) DRAM buffer + PRAM main memory

Fig. 1. Two types of hybrid main memory architectures with DRAM and PRAM.

applications, there is a need for increased main memory capacity. However, with the increasing memory requirement, DRAM memory systems have been faced with practical limitations in terms of high power consumption and low scalability. For example, the DRAM idle power has significantly increased in proportion to its size, which has a substantial effect on total power consumption [Roberts et al. 2009]. Also, DRAM has almost reached its capacity limit because of the scalability wall [Ipek et al. 2010].

As a remedy to the limitations, many researchers have attempted to replace DRAM with Phase Change RAM (PRAM), an emerging memory technology, as an alternative main memory device. PRAM has a nonvolatile characteristic, thereby having little idle power consumption. It is also believed that PRAM is more scalable than DRAM [Ipek et al. 2010]. Therefore, using PRAM as the main memory can increase the memory capacity. However, the primary challenges in deploying PRAM include its latency and endurance issues.

From an architectural perspective, two hybrid main memory design options have been proposed to solve the latency issue of PRAM as shown in Figure 1. Figure 1(a) shows the configuration that locates DRAM and PRAM in the same linear region [Dhiman et al. 2009; Park et al. 2010, 2011b; Ramos et al. 2011]. In this configuration, data in both DRAM and PRAM can be directly accessed, and a conventional memory controller can be used without any changes. In order to solve the endurance problem, data allocation and migration schemes in the OS layer, as well as additional hardware components, are needed. Figure 1(b) describes another configuration that locates the DRAM in front of the PRAM [Park et al. 2011a, 2012; Qureshi et al. 2009a, 2009b; Lee

et al. 2009; Ferreira et al. 2010]. This configuration can solve the endurance problem better than the first configuration thanks to the DRAM buffer. Consequently, we incorporated the second configuration of PRAM main memory with a DRAM buffer into our system as an underlying memory architecture.

In order to solve the endurance problem of PRAM main memory with a DRAM buffer, various wear-leveling algorithms have been presented [Dhiman et al. 2009; Qureshi et al. 2009a, 2009b; Lee et al. 2009; Ferreira et al. 2010; Park et al. 2012]. There are two wear-leveling algorithm goals: write count reduction and uniform write count distribution. In order to reduce the PRAM write count, the DRAM buffer kept frequently updated data [Qureshi et al. 2009b; Ferreira et al. 2010; Park et al. 2012], and data comparison write schemes were used for writing only modified bits/lines [Lee et al. 2009]. In order to even out the PRAM write count, swapping and shifting schemes were presented [Dhiman et al. 2009; Qureshi et al. 2009a, 2009b; Ferreira et al. 2010; Park et al. 2012].

While previous wear-leveling algorithms increased the PRAM lifetime, we identified three critical improvement points by thoroughly investigating them. First, previous DRAM buffering schemes did not account for write count distribution. They were only designed for write count reduction. Consideration of PRAM's endurance in the DRAM buffer helps to even out the PRAM write count as well as reduce the PRAM write count. Second, the swapping and shifting operations have parameters: operation timing and the number of swapping pages. Careful attention should be paid to these parameters because they critically affect wear-leveling. With static parameters, it is hard to achieve good wear-leveling. However, the wear-leveling effect can be improved if the parameters are adaptively controlled based on the workload pattern. Finally, the swapping and shifting operations were loosely coupled with data in the DRAM buffer. The cooperation between the data in the DRAM buffer and the swapping and shifting operations in the PRAM main memory can alleviate the overhead incurred by additional operations.

Motivated by these observations, we developed an adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer. As depicted in Figure 1(b), we developed three novel schemes: a PRAM-aware DRAM buffering scheme, an adaptive multiple data swapping and shifting scheme, and a DRAM buffer-aware swapping and shifting scheme. The following points discuss the schemes in more detail.

—Our PRAM-aware DRAM buffering scheme considers the PRAM write count and clean data based on the LRU scheme for victim selection when the DRAM buffer is full. It can prevent the skewing of the write count of specific pages, as well as reduce the PRAM write count, thereby improving the PRAM lifetime.
—To improve the PRAM endurance, the PRAM write count needs to be evened out. An adaptive multiple data swapping and shifting scheme was designed to accomplish this. Swapping multiple pages at once can efficiently even out the write count among all the pages. A shifting scheme was also applied to even out the write count among all the lines in each page. To effectively perform the swapping and shifting operations, the operation timing and number of swapping pages are dynamically controlled based on the workload pattern. The adaptive multiple data swapping and shifting scheme can prolong the PRAM lifetime.
—In order to alleviate the overhead incurred by additional swapping and shifting operations, a DRAM buffer-aware swapping and shifting scheme was proposed. The DRAM buffer contents are considered during the swapping and shifting operations, thereby reducing unnecessary write operations.

This study is an extension of our previous work [Park et al. 2012], in which we focused on the basic buffering, swapping, and shifting designs for PRAM main memory

with a DRAM buffer. Our objective in this study, however, was to devise a full-fledged wear-leveling scheme for PRAM main memory with a DRAM buffer, and transparently integrate the overall components in a PIN-based wear-leveling simulator.

The remainder of this article is organized as follows. The article begins with background about PRAM, and introduces previous wear-leveling algorithms in Section 3. Section 4 describes an adaptive wear-leveling algorithm that consists of a PRAM-aware DRAM buffering scheme, an adaptive multiple data swapping and shifting scheme, and a DRAM buffer-aware swapping and shifting scheme. The evaluation is explained in Section 5. Section 6 concludes the article and presents future work.

2. BACKGROUND

The purpose of this section is to provide background for our work. We introduced a PRAM cell structure, and discussed the advantages and disadvantages of PRAM.

Many researchers have developed new memory technologies (Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM)). Among these memories, PRAM is considered as the most promising technology for future memory. PRAM uses a phase change material (GST: Ge2Sb2Te5), which has two phases: an amorphous or a crystalline phase. Since the amorphous and the crystalline phases have a large variance in their resistance, the data are read by measuring the current of the PRAM. The phase of the GST can be changed by heating the material. A moderate and long current pulse crystallizes GST. On the other hand, a short current pulse melts and quenches GST quickly, and makes it amorphous.

Basically, PRAM is byte-addressable like DRAM. The great advantages of PRAM are high scalability and low idle power consumption [Dhiman et al. 2009]. PRAM does not require an implementation of a capacitor for memory cells, thus providing superior density relative to DRAM. Because the phase of PRAM is maintained persistently, PRAM has negligible leakage power consumption. Therefore, PRAM can provide much higher capacity and lower power consumption than DRAM.

However, the long current pulse for crystallizing increases the PRAM write latency. Although the PRAM access latency is tens of nanoseconds, it is still not comparable to the DRAM access latency. The frequent access of PRAM can impact the performance. The high write power consumption for the long pulse for phase change and write endurance are also limitations of PRAM [Dhiman et al. 2009]. PRAM writing makes the material thermally expand and contract. It degrades the electrode-storage contact and reduces the reliability of the programming current. Thus, the write endurance of PRAM cells is degraded. PRAM can sustain 10^8 rewrites per cell [Zhou et al. 2009; Lee et al. 2009; Zhang and Li 2009; Kong and Zhou 2010].

3. RELATED WORK

Many wear-leveling algorithms have emerged to overcome the PRAM endurance problem as main memory or storage. These algorithms are categorized into two design issues: write count reduction and uniform write count distribution, as described in Table I. In this section we will present a careful analysis of previous wear-leveling algorithms.

3.1. Write Count Reduction

The redundant write removal and DRAM buffering algorithms are the main ones used to reduce the PRAM write count. In this section, we will describe them in detail and introduce other methods like encryption and content-aware block placement.

3.1.1. Redundant Write Removal. Yang et al. [2007] proposed a data-comparison write (DCW) scheme. This scheme reads the stored data during the write operation, and

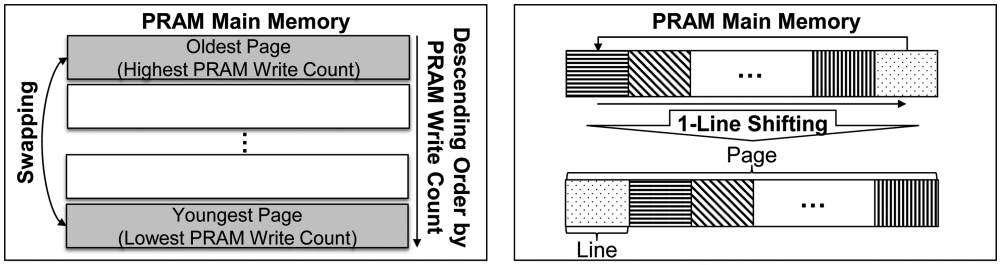
Table I. Related Wear-Leveling Categorization

Write Count Reduction		Uniform Write Count Distribution	
Redundant Write Removal	DCW [Yang et al. 2007] Partial Writes [Lee et al. 2009] Redundant Writes [Zhou et al. 2009] RWR [Ferreira et al. 2010] Flip-N-Write [Cho and Lee 2009] LLWB [Qureshi et al. 2009b] Sub-Page [Ferreira et al. 2010]	Data Swapping	Page Swapping [Dhiman et al. 2009] Segment Switching [Park et al. 2008] Segment Swapping [Zhou et al. 2009] Swap Algorithm [Ferreira et al. 2010] Data Swapping [Park et al. 2012] Wear Rate Leveling [Dong et al. 2011]
DRAM Buffering	Lazy Write [Qureshi et al. 2009b] N-Chance [Ferreira et al. 2010] Line-Level LRU [Park et al. 2012]	Data Shifting	Word-Level Shifting [Park et al. 2008] Row Shifting [Zhou et al. 2009] FGWL [Qureshi et al. 2009b] Start-Gap [Qureshi et al. 2009a] Data Shifting [Park et al. 2012]
Other Methods	ADCW [Zhang and Li 2009] Encryption [Kong and Zhou 2010] CA [Wongchaowart et al. 2010]	Other Methods	PA-to-PCMA translation [Seznec 2010]

then writes the input data only when the input and stored data are different. Lee et al. [2009] proposed a partial write scheme that reduces the number of writes to PRAM by tracking the dirty data from the L1 cache to the memory banks. Zhou et al. [2009] proposed a redundant bit-write scheme that removes unnecessary bit-writes during an entire row (or a page) update. Ferreira et al. [2010] proposed a page partitioning scheme to avoid the write-back of clean subpages, and a read-write-read (RWR) scheme to remove unnecessary writes within a subpage. In order to improve the DCW scheme, Cho and Lee [2009] proposed a flip-N-write scheme that updates, at most, N/2 bits at a time when updating N bits by exploiting read-modify-write and data encoding schemes. Qureshi et al. [2009b] proposed a line level writeback (LLWB) scheme to write only dirty lines within a page. By reducing wasteful writes, these schemes can improve PRAM endurance. However, they have drawbacks in that they require additional read operations and storage overhead to track the dirty data.

3.1.2. DRAM Buffering. As another method to reduce the PRAM write count, the DRAM buffer is used to maintain frequently updated data. Qureshi et al. [2009b] proposed the lazy-write organization, which avoids the first write to PRAM for the dirty pages. Ferreira et al. [2010] proposed an N-chance cache replacement mechanism, which selects a victim from a page set, the oldest clean page among the N least recently used pages, for writeback minimization. Park et al. [2012] proposed a line-level LRU scheme to reduce the PRAM write count. This scheme needs no additional bits to identify dirty lines because the request is the same as the line size. Although these DRAM buffering schemes can improve PRAM endurance by reducing the PRAM write count, better PRAM endurance improvement can be achieved by considering the uniform write count distribution.

3.1.3. Other Methods. Zhang and Li [2009] proposed an adaptive data comparison write (ADCW) scheme that eliminates redundant bit-writes to PRAM cells that require substantially higher programming due to Process Variation (PV). Kong and Zhou [2010] presented simple, yet effective extensions to the encryption scheme to revive partial writes. Wongchaowart et al. [2010] presented a content-aware block placement scheme to reduce the number of PRAM bit programming operations. When a block write request is received, it is located within free blocks that are similar to a given data block.



(a) Page-level swapping operation (b) Line-level shifting operation

Fig. 2. Basic swapping and shifting operations.

In order to quickly locate a new block of data within a free block with similar contents, free blocks are indexed by using a content-based signature.

3.2. Uniform Write Count Distribution

To even out the write count in the PRAM main memory, various swapping and shifting schemes were used. We will explain them in detail in the following.

3.2.1. Data Swapping. The basic swapping operation is to periodically exchange data in the oldest page (or segment), which means the most-written page among all the pages, with data in the youngest page, which means the least-written page among all the pages as shown in Figure 2(a). Thus, the write count can be evened out among all the pages. Dhiman et al. [2009] proposed a page swapping scheme that moves all the free pages from the threshold-free list to the original free list. This occurs if the number of writes to any PRAM page exceeds a given threshold. Park et al. [2008] proposed a segment switching scheme that swaps the data in the oldest segment with data in the youngest segment. The segment switching is performed if the difference between the write count of the oldest segment and the youngest segment is larger than a threshold value. Zhou et al. [2009] also proposed a segment swapping scheme with a large segment and a *last_swapped* parameter. The large segment reduced the storage overhead, and the *last_swapped* parameter prevented the segment from being selected again too soon. Ferreira et al. [2010] proposed a PCM-aware swap algorithm that swaps pages on page cache writebacks. Park et al. [2012] designed a multiple data swapping scheme. This scheme swaps data in the oldest pages with data in the youngest pages in order to efficiently handle a wide range of the PRAM main memory. Dong et al. [2011] proposed a wear rate leveling mechanism that considers the wear rates (write traffic/endurance) by migrating hot/cold data to strong/weak domains. A maximum hyperweight rematching scheme was also proposed to solve a trade-off between the endurance improvement and the swapping data volume. While these swapping schemes can achieve uniform write count distribution among the pages, they have the limitation of being statically performed despite the dynamic workload pattern.

3.2.2. Data Shifting. While the swapping operations even out the write count among all the pages, evening out the write count among the lines in each page is needed. The basic shifting operation is to periodically store lines of each page in a rotated manner as shown in Figure 2(b). Park et al. [2008] proposed a word-level shifting scheme that stores the segments after shifting several words of the data from the beginning of the segments. This can prevent the specific usage pattern in the virtual segment being repeated in other physical segments. Zhou et al. [2009] proposed a row shifting scheme by one byte on every 256 writes, which shows the most effective result. Qureshi et al. [2009a] proposed a start-gap scheme, which moves one line from its

location to a neighboring location with two registers, start and gap. The gap register keeps track of how many lines have moved, and the start register is incremented to keep track of the number of times all lines have moved. With these two registers, the start-gap scheme can achieve low storage overhead for implementing the shifting operation. Park et al. [2012] presented a data shifting scheme that considers a shifting level and shifting period. Because the shifting schemes are also statically performed, their wear-leveling effect could be improved if the operation timing is dynamically changed.

3.2.3. Other Methods. As another method to even out the write count, Seznec [2010] presented a PA (the physical memory address)-to-PCMA (the PRAM memory address) translation scheme to protect from malicious overwrite by evening out and randomizing the write flow of the PRAM main memory. The PA-to-PCMA translation is continuously modified through a random process. This scheme can also be applied to conventional nonmalicious applications.

3.3. Additional Improvement Issues

While previous studies have increased the PRAM lifetime by designing various wear-leveling algorithms (redundant write removal, DRAM buffering, data swapping, and data shifting), there are several additional improvement issues related to the wear-leveling effect. First, previous DRAM buffering schemes only focused on reducing the PRAM write count, but a better wear-leveling effect can be achieved by considering the write count and clean data. Skewed writes to specific pages can be prevented, by considering the write count, thereby improving the PRAM lifetime. By applying the clean first scheme [Park et al. 2006; Ou et al. 2009], the PRAM write count can be reduced by making room for the dirty data. Therefore, we created a PRAM-aware DRAM buffering scheme, thus reducing the write count and preventing skewed writing. Second, previous swapping and shifting operations were performed on every N writes, even though the workload pattern was changing dynamically. The wear-leveling effect could be improved with dynamic operation timing. We developed an adaptive multiple data swapping and shifting scheme, thus making the write count even among all the pages and lines. Finally, during swapping and shifting operations, exploiting a DRAM buffer can also improve the wear-leveling effect. Without considering the DRAM buffer in the swapping and shifting operations, unnecessary write operations would be allowed. Therefore, the PRAM lifetime was improved by exploiting the dirty data in the DRAM buffer. This scheme was inspired by BAGC [Lee et al. 2008], which reduces unnecessary page migrations during a garbage collection operation in flash memory-based storage systems. By designing a DRAM buffer-aware swapping and shifting scheme, we reduce unnecessary writes by curbing additional swapping and shifting operations. Through an adaptive wear-leveling algorithm that consists of a PRAM-aware DRAM buffering scheme, an adaptive multiple data swapping and shifting scheme, and a DRAM buffer-aware swapping and shifting scheme, a better wear-leveling effect as compared to previous algorithms, can be achieved.

4. ADAPTIVE WEAR-LEVELING ALGORITHM

We proposed an adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer in order to improve PRAM endurance by reducing and evening out the PRAM write count. To efficiently even out the PRAM write count among all the pages and lines, we created an adaptive multiple data swapping and shifting scheme. By applying dynamic operation timing to the swapping and shifting operations, PRAM endurance can be improved. Being aware of the DRAM buffer can also reduce the overhead incurred by additional swapping and shifting operations.

4.1. PRAM-Aware DRAM Buffering Scheme

In the hybrid main memory that locates DRAM in front of PRAM, DRAM is used as a buffer to reduce latency and improve the endurance of the PRAM main memory. A relatively small DRAM buffer (3% of the size of the PRAM main memory) can bridge most of the latency gap between DRAM and PRAM [Qureshi et al. 2009b]. Due to the DRAM buffer size limitation, it is necessary to secure the DRAM buffer space by designing a novel victim selection algorithm. When designing the victim selection algorithm, uniform write count distribution, as well as write count reduction, should be considered.

4.1.1. Design Issues. The main goal of the DRAM buffering scheme is to reduce the PRAM write count. To achieve this, many previous studies have used a least recently used (LRU) scheme that increases the hit ratio by evicting the least recently used data first. This is because the LRU scheme exploits a temporal locality, meaning that recently accessed data will be referenced again in the near future. Therefore, the PRAM write count can be reduced by keeping frequently updated data in a DRAM buffer for a long time. We also considered the clean data to reduce the PRAM write count. The clean data are referenced, but not modified, while the dirty data are modified by the write operation. Thus, only dirty data increases the PRAM write count. In the case of a LRU-only scheme, the dirty data may be evicted to the PRAM main memory, thereby resulting in an increased write count. Therefore, considering clean data is important when designing a DRAM buffering policy. Although previous DRAM buffering policies have focused on reducing the PRAM write count, in our design, uniform write count distribution can be achieved by considering the PRAM write count. Since the memory controller handles both the DRAM buffer and PRAM main memory, a DRAM buffering scheme that considers the PRAM write count can be created. This can prevent the skewing of the write count of a specific page.

4.1.2. Algorithm Description. By investigating the temporal locality, clean data, and PRAM write count, a PRAM-aware DRAM buffering scheme was created, as shown in Figure 3. The DRAM buffer was divided into two layers for the application of different policies to each layer. We applied the least recently used (LRU) scheme to both clean and dirty data in the first layer of the DRAM buffer. This discards the least recently used data first from the first layer because the data are considered "cold data" if not frequently updated. In this way, the LRU scheme can reduce the PRAM write count by maintaining frequently updated data in the DRAM buffer. In the second layer, the clean data and write count are used for the victim selection. In this layer, the clean data are evicted first to make room in the DRAM buffer as well as reduce the PRAM write count. If there are no clean data in the second layer, the write counts of dirty data in the second layer are compared and the dirty data with the lowest write count are chosen. By using the address as the key, the write counts in the PRAM main memory can be measured. By considering the write count, we can prevent the write count skewing of a specific page. This results in reducing the maximum write count, thus the PRAM lifetime increases. If the dirty data have the same write count, the data in the LRU location are selected as a victim. We adopted a line-level granularity as the management granularity because the request size from the memory controller is equal to the line size of the last level cache.

4.1.3. Window Size. In the PRAM-aware DRAM buffering scheme, it is also important to determine the size of the second layer, called window size, because it critically affects both the wear-leveling effect and performance overhead. If the window size is large, the PRAM-aware DRAM buffering scheme can improve the wear-leveling effect by working with the write count and clean data, but may incur performance overhead due to less

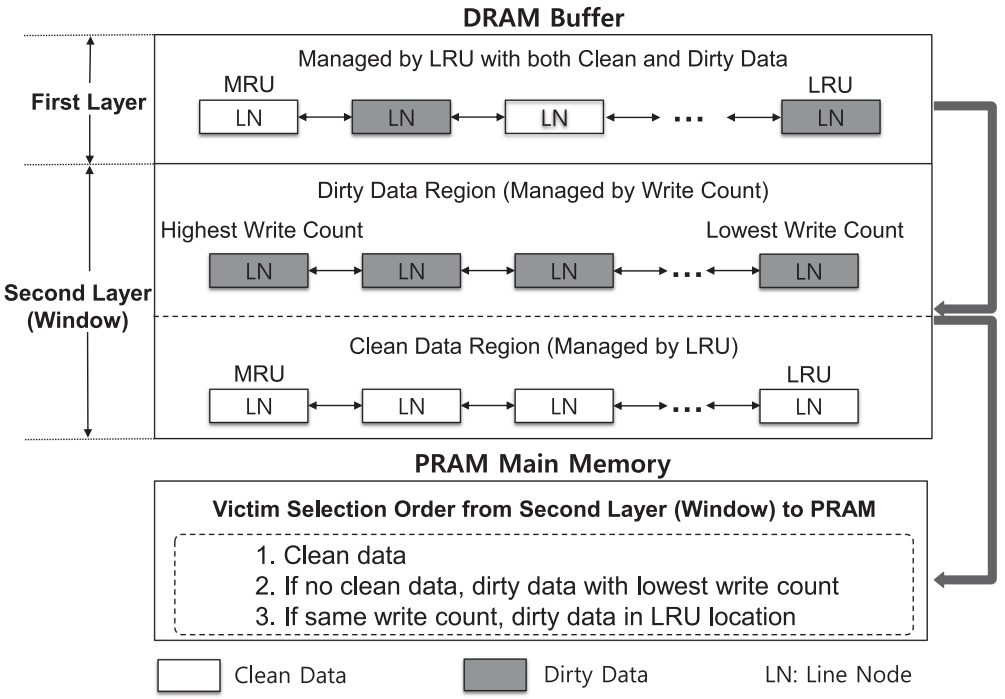


Fig. 3. PRAM-aware DRAM buffering scheme.

efficiency in keeping frequently updated data via the LRU scheme and a large sorting overhead in the window. However with a small window size, we cannot achieve the wear-leveling effect with the write count and clean data. In order to determine an appropriate window size, we performed the evaluation with various window sizes in Section 5.

4.2. Adaptive Multiple Data Swapping and Shifting Scheme

In order to improve the PRAM endurance, it is also important to even out the write count among all the lines and pages of the PRAM main memory. To achieve this, an adaptive multiple data swapping and shifting scheme was developed. In order to improve the wear-leveling effect, parameters such as the operation timing and the number of swapping pages, were considered. In the following section, the concept and benefits of the proposed swapping and shifting scheme will be explained in detail.

4.2.1. Motivation. Previous swapping schemes [Dhiman et al. 2009; Park et al. 2008; Zhou et al. 2009; Ferreira et al. 2010; Park et al. 2012] periodically exchange the data of the oldest page with data from the youngest page to even out the write count among all the pages. Previous shifting schemes [Park et al. 2008, 2012; Zhou et al. 2009; Qureshi et al. 2009a, 2009b] also periodically shift data in each page to even out the write count among all the lines. However, previous swapping and shifting schemes are limited due to their periodic operation and one-page swapping.

4.2.2. Design Issues. The operation timing and the number of swapping pages should be carefully considered when designing swapping and shifting schemes because they critically affect the wear-leveling effect. First, when determining the operation timing, we have to account for the workload pattern. If the write accesses are skewed to a specific page or line, frequent swapping and shifting operations are efficient. However, no operation is very efficient if the write accesses are evenly distributed to the PRAM

main memory because the swapping and shifting operations consist of read and write operations, thereby increasing the PRAM write count. Therefore, it is necessary to change the operation timing dynamically depending on the workload pattern.

Second, the number of pages that the swapping operation can perform at once is important to consider. Previous swapping schemes swapped only one page at a time. Although one-page swapping is efficient if a specific page is skewed intensively, most workload patterns access a wide region of the PRAM main memory. Thus, it is hard to achieve the wear-leveling effect with one-page swapping. To overcome this limitation, some previous studies expanded the page size, but they created uneven write accesses on a page. Therefore, it is necessary to design the swapping operation to-efficiently cover a wide range of PRAM main memory.

4.2.3. Algorithm Description. By considering the operation timing and the number of swapping pages, we proposed an adaptive multiple data swapping and shifting scheme. To adapt the workload pattern dynamically, we periodically monitored the increase in the maximum write count, which represents whether the write access is skewed or not. The maximum write count denotes the maximum page write count among all the pages of the PRAM main memory for the swapping operation, and the maximum line write count among all the lines in each page for the shifting operation. Based on this information, we can predict the future workload pattern. As a prediction method, we used the exponentially weighted moving average (EWMA) [Ye et al. 2004], which is calculated by (1). In EWMA, N means a check period that is measured by the number of writes to the PRAM main memory, and n^*_{t+1} represents an increase in the maximum write count in the future. Thus, if EWMA is over a threshold value, we regarded the write access as being skewed.

$$n^*_{t+1} = \alpha n_t + (1-\alpha) n^*_t$$
$$(\alpha = 2/N + 1)$$

, where

n^*_{t+1}: EWMA of next check period,
n_t: max. write count of current check period,
n^*_t: EWMA of current check period,
N: check period size. (1)

To efficiently cover a wide range of the PRAM main memory, instead of using a large page (1MB), we chose multiple 4KB pages for the swapping operation. The number of swapping pages is determined by the interval between swapping operations. The interval is evaluated by the number of writes to the PRAM main memory. If the interval is short, a small range of the PRAM main memory may be accessed. Thus, only a few pages are selected for the swapping operation. On the contrary, if the interval is long, many pages are selected to cover a broad range of the PRAM main memory. To do this, we can also minimize the overhead incurred by the swapping operations. In this article, the overhead was set to under 1%. Therefore, the ratio of the swapping pages to the total PRAM write counts cannot exceed the 1% predefined overhead. By swapping multiple oldest pages with multiple youngest pages at once, the write count among all the pages of the PRAM main memory may be efficiently evened out. This produces a better wear-leveling effect compared to the scheme that handles swapping only one page at a time. A line-level data shifting scheme, which shifts a page at a line level because the request size from a last level cache matches the line size, was also designed. This evens out the write count among all the lines within a page.

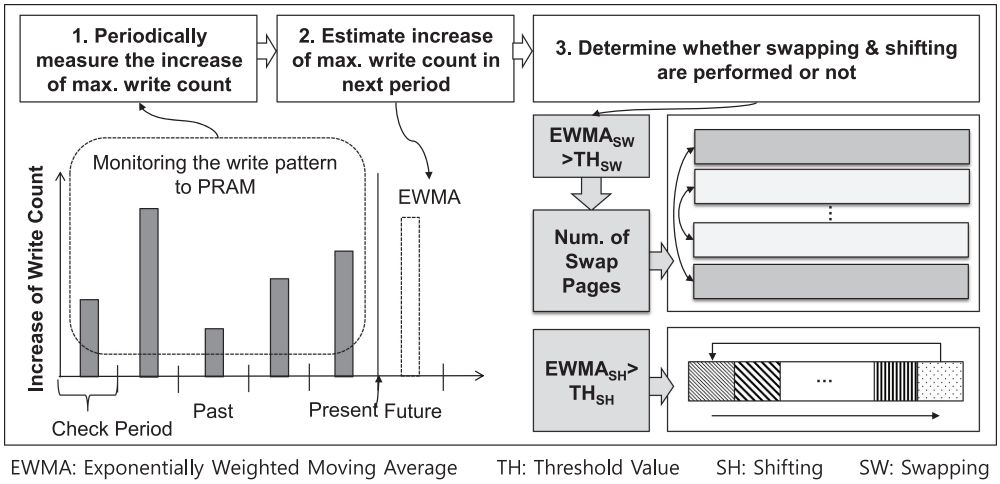


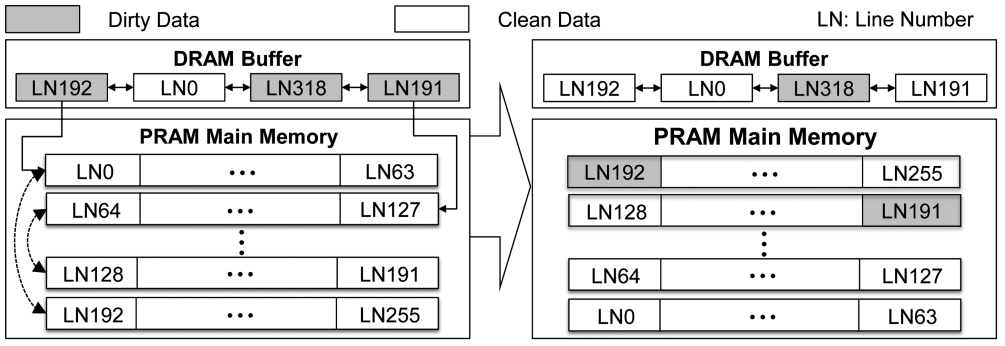
Fig. 4. Operation flow of adaptive multiple data swapping and shifting scheme.

4.2.4. Operation Flow of Adaptive Multiple Data Swapping and Shifting. Figure 4 shows the operation flow of an adaptive multiple data swapping and shifting scheme. For the swapping operation, the increase in the maximum page write count among all the pages in the check period is monitored first. The check period for the swapping operation is determined by the number of write accesses to the PRAM main memory. Using this value, the increase in the maximum page write count, EWMA_{SW}, can be estimated for the next future period. If the expectation value, EWMA_{SW}, is over a threshold value, TH_{SW}, the number of swapping pages is determined. Then, we find the old and young pages by sorting the write counts in the PRAM main memory. Finally, the swapping operations between those pages are performed. The shifting operation is similar to the swapping operation. First, the increase in the maximum line write count among all the lines of each page in the check period is monitored. The check period for the shifting operation is determined by the number of write accesses to each page. The number of write accesses to each page when a write request is issued to the PRAM main memory is easily known. The increase in the maximum line write count, EWMA_{SH}, in the next future period can also be estimated. If EWMA_{SH} is over a threshold value, TH_{SH}, a shifting operation is performed. If not, no operation is performed because the write accesses are evenly distributed. Via the adaptive multiple data swapping and shifting scheme, the wear-leveling effect of basic swapping and shifting schemes can be improved, thereby reducing the maximum write count and evening out the PRAM write count.

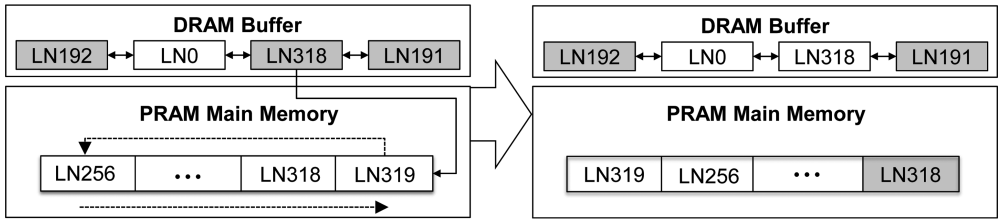
4.3. DRAM Buffer-Aware Swapping and Shifting Scheme

In order to reduce the additional write overhead, the data in the DRAM buffer should be considered during the swapping and shifting operations. In this section, a DRAM buffer-aware swapping and shifting scheme will be explained in detail.

4.3.1. Motivation. There are two reasons to increase the PRAM write count: dirty data eviction from the DRAM buffer and writes by the swapping/shifting operation. We found that there are duplicate writes between dirty data eviction and the swapping/shifting operation. This results in increasing the PRAM write count. The dirty data in the DRAM buffer are more up-to-date than the data in the PRAM main memory. Therefore, if the data in the DRAM buffer are not considered when swapping and shifting



(a) DRAM Buffer-aware multiple data swapping



(b) DRAM Buffer-aware Line-level data shifting

Fig. 5. Example of buffer-aware swapping and shifting.

operations are performed, the PRAM write count will increase due to the swapping and shifting operations, as well as the dirty data evicted from the DRAM buffer.

4.3.2. Algorithm Description. To overcome the unnecessary write overhead, a DRAM buffer-aware swapping and shifting scheme was proposed. By reducing the PRAM write count, the PRAM endurance can be improved. This scheme uses the dirty data in the DRAM buffer. If there are data whose line number (LN) is contained in the selected pages for the swapping and shifting operations, the data are simultaneously evicted during the operations. Then the line is marked as clean data because the data are reflected to the PRAM main memory. By doing this, we can maintain a hit ratio in the DRAM buffer as well as reduce the PRAM write count.

Figure 5 shows a DRAM buffer-aware data swapping and shifting scheme. In Figure 5, the gray box represents the dirty data, while the white box represents the clean data. LN is the line number, and we assumed a page consists of 64 lines. During the swapping operation, the page that contains lines from LN0 to LN63 is interchanged with the page that contains lines from LN192 to LN255. In this case, LN192 in the DRAM buffer is selected for the swapping operation because it is dirty and contained in the exchanged pages. Thus, LN192 in the DRAM buffer is simultaneously evicted to the PRAM main memory during the swapping operation. Then it is marked as clean data in the DRAM buffer, as shown in Figure 5(a). In the case of LN191, its operation is the same as that of LN192. Therefore, unnecessary writes to the PRAM main memory can be reduced. During the shifting operation, 1-line shifting of a page is performed, which consists of lines from LN256 to LN319. LN318 is located in the DRAM buffer as dirty data. Similar to the DRAM buffer-aware swapping operation, LN318 is simultaneously evicted to the PRAM main memory during the shifting operation. After the shifting operation, this line is marked as clean data as shown in Figure 5(b).

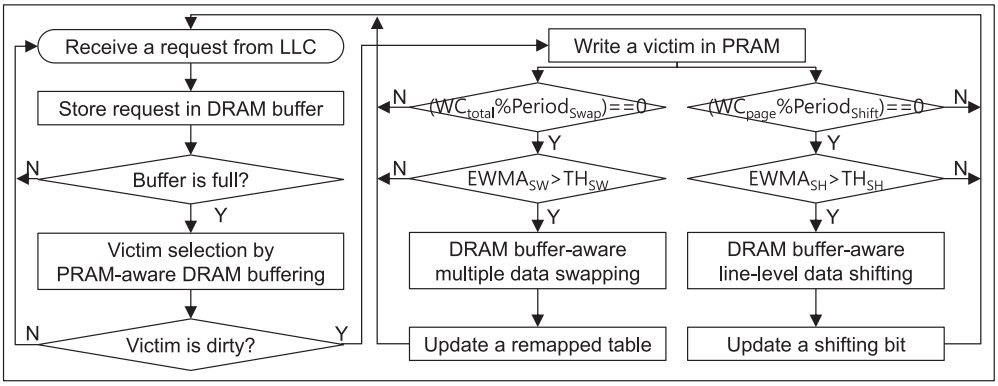


Fig. 6. Flow chart of adaptive wear leveling algorithm.

By exploiting data in the DRAM buffer during swapping and shifting operations, the additional write overhead can be reduced, thereby improving the PRAM endurance.

4.4. Overall Operation Flow

The flow chart of the adaptive wear-leveling algorithm is depicted in Figure 6. This operation starts when a memory controller receives a request from a last level cache (LLC). The request is stored in the DRAM buffer and managed by the PRAM-aware DRAM buffering algorithm. If the DRAM buffer is full due to its limited size, a victim is selected for securing a DRAM buffer space, as explained in Section 4.1. If the victim is clean, there is no need to write it because it is already stored in the PRAM main memory. Otherwise, the victim is written to the PRAM main memory.

The dirty victim, which is evicted from the DRAM buffer, is written to the PRAM main memory, thereby increasing the PRAM write count. The total PRAM write count is checked for the swapping operation, while the page write count is checked for the shifting operation. Using the write counts, EWMA values are calculated for identifying skewed write accesses, as explained in Section 4.2. If this meets the conditions, a DRAM buffer-aware multiple data swapping or shifting is performed. These swapping and shifting operations can be performed independently. After the swapping and shifting operations, the write counts, remap table, and shifting bits, which are stored in the PRAM main memory, are finally updated.

4.5. Implementation Details

To support the adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer, we designed a new memory controller and hybrid main memory, as shown in Figure 7. We implemented additional hardware components, which are the DRAM buffer manager, swapping and shifting module, address remap table, and write count manager, in the conventional memory controller [Ramos et al. 2011].

The role of the DRAM buffer manager is to control the DRAM buffer like a cache, which is managed by the PRAM-aware DRAM buffering scheme. A capacitor is used to protect the data in the DRAM buffer at a sudden power off [Guo et al. 2013]. Under the backup power, the data in the DRAM buffer are moved to storage. The swapping and shifting module performs the swapping and shifting operations in the PRAM main memory. The write count per each line is managed in the write count manager. While managing the line write count, we need a page write count for the PRAM main memory whose management unit is a page. Therefore, we calculated the page write count by using the line write counts. A page consists of several lines. Thus, we defined the

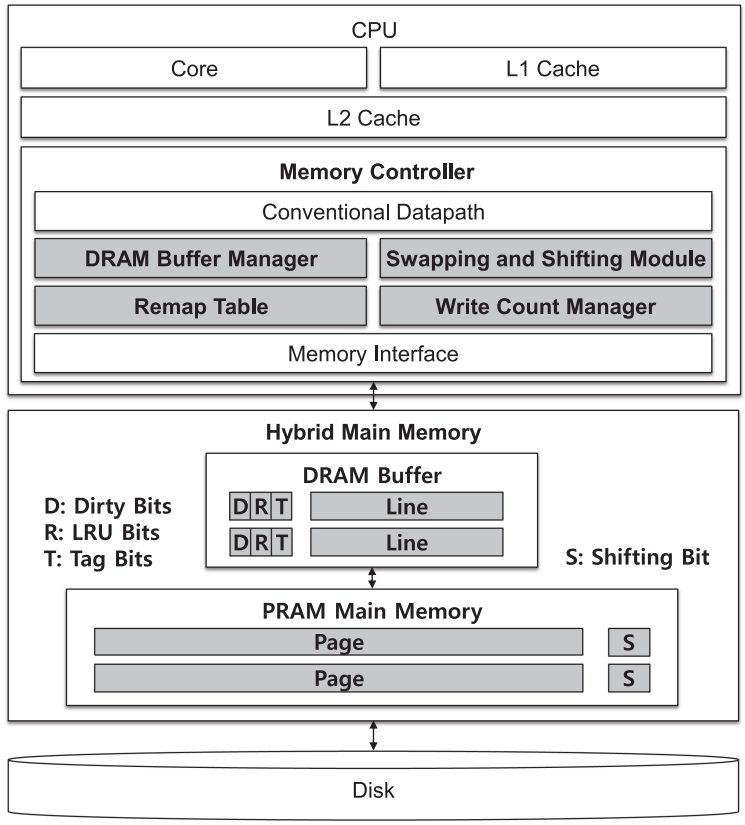


Fig. 7. Overall architecture for the adaptive wear-leveling technique.

page write count as the highest line count among the lines in a page. A remap table was designed to support the swapping scheme. The swapping scheme exchanges data between two physical pages, thereby making the remap table necessary. When the request arrives at the memory controller, the remap table is first checked to determine whether or not the physical address has changed. Then that request is served.

The dirty bits, LRU bits, and tag bits are made of SRAM for fast latency. The write counts, remap table, and shifting bits are stored in the PRAM main memory, and periodically written to storage in order to prepare for power failure [Qureshi et al. 2009b]. During the reboot, they are first stored before the system shuts down. Then, when the system starts up, the space in the PRAM main memory is reserved, and the data are copied from storage [Dhiman et al. 2009].

4.6. Overhead Analysis

We summarized a storage overhead for managing a 16MB DRAM buffer and 512MB PRAM main memory, as described in Table II. For the DRAM buffering scheme, an additional storage overhead is required for tag bits, dirty bits, and LRU bits, which are stored in SRAM. Forty-five bits of tag-store (26 tag bits + 1 dirty bit + 18 LRU bits) for each line whose size is 64B, are needed. Thus, 1.4MB is necessary for a 16MB DRAM buffer. In order to implement an adaptive wear-leveling algorithm, the storage overhead is also needed to house the write count, remap table, and shifting bits, which are stored in PRAM. First, the proposed wear-leveling schemes are based on the write

Table II. Storage Overhead for Managing 16MB DRAM Buffer with 512MB PRAM Main Memory

	Structure	Storage Overhead	Storage Location
Metadata for DRAM Buffer (16MB)	Tag Bits	832KB (26bits/64B)	SRAM
	Dirty Bits	32KB (1bit/64B)	
	LRU Bits	576KB (18bits/64B)	
Metadata for PRAM Main Memory (512MB)	Write Counts	32MB (4B/64B)	PRAM
	Remap Table	512KB (4B/4KB)	
	Shifting Bits	96KB (6bits/4KB)	
Total Storage Overhead		34MB	

count per line unit. Therefore, 32MB for 512MB PRAM main memory is needed because 4B is needed for storing each line's write count. 4B is large enough to handle over 10^8 write counts. Second, a remap table is needed for the swapping operation. In order to maintain the remap table, we need 0.5MB for 512MB PRAM main memory because 4B is needed to store each page's address. For the shifting operation, 6 bits per page is needed, thereby making 96KB necessary for 512MB of PRAM main memory. Therefore, the total storage overhead for managing the 16MB DRAM buffer and 512MB PRAM main memory, is 34MB. Except for the write count management, the storage overhead is 2MB which takes 0.4% of the total main memory size. In order to reduce the write count management overhead, we will consider bit arrays [Chang et al. 2010]. By using a clock algorithm instead of an LRU scheme, we can also reduce the storage overhead. With these schemes, the total storage overhead can be significantly reduced.

The adaptive wear-leveling algorithm is accompanied by the performance overhead. First, in the PRAM-aware DRAM buffering scheme, the victim selection procedure needs to find a victim with the lowest write count in the window, which requires a linear search. Although the linear search has O(n) complexity, it cannot significantly influence the overall performance because of a limited window size [Jo et al. 2006]. Second, the swapping and shifting operations consist of read and write operations, thereby potentially creating a latency problem, especially in a multiple swapping operation that exchanges multiple pages at once. In order to overcome the latency problem in the future, exploiting the write pausing and cancellation scheme [Qureshi et al. 2010] or the bank usage table scheme with the multibank characteristic [Delaluz et al. 2002] will be considered. Therefore, we can improve the latency by reducing the overhead incurred by wear-leveling operations. It is better to show the actual measurement of the processing and time overhead, but the current memory controller is designed for measuring the PRAM write counts to show the wear-leveling effect. In the future we will analyze this computation overhead with a cycle-accurate memory controller.

5. EXPERIMENT

In this section, we present the experimental environment and results obtained with the adaptive wear-leveling algorithm. In the experimental environment section, the specifications of the simulator, workload characteristics, metrics, and comparisons are explained. In the experimental results section, the effects of the PRAM-aware DRAM buffering scheme, adaptive multiple data swapping and shifting scheme, and DRAM buffer-aware data swapping and shifting scheme as compared to previous wear-leveling algorithms are shown. Finally, the PRAM lifetime is discussed.

5.1. Experimental Environment

In order to evaluate an adaptive wear-leveling algorithm, a simulator based on PIN was implemented [Luk et al. 2005]. As shown in Figure 8, a function to analyze the memory accesses was used. We also modeled the DRAM buffer and PRAM main memory, as

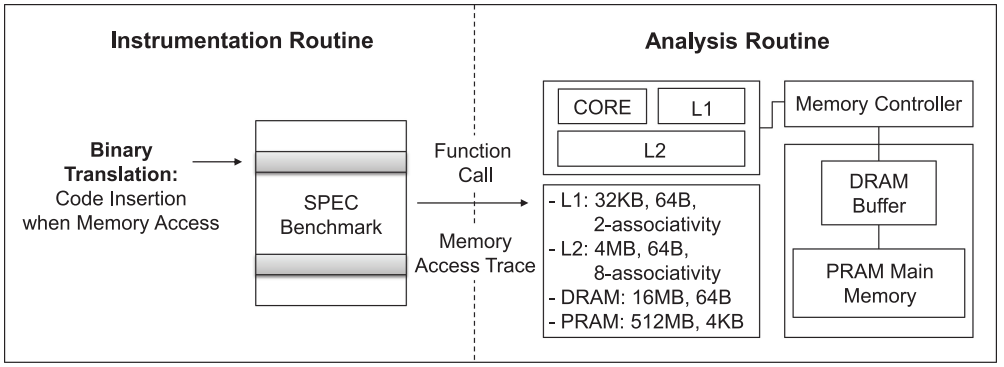


Fig. 8. Overall architecture of PIN-based wear-leveling simulator.

Table III. SPEC CPU2006 Benchmark Description[1]

Benchmarks	Description
bzip2	Compression
gcc	C Compiler
mcf	Combinational Optimization
gobmk	Artificial Intelligence: go
sjeng	Artificial Intelligence: chess
libquantum	Physics: Quantum Computing
h264ref	Video Compression
omnetpp	Discrete Event Simulation

well as the memory controller, and L1 and L2 caches. The L1 and L2 caches, as well as the DRAM buffer, all have 64B blocks with capacities of 32KB and two-associativity, 4MB and eight-associativity, and 16MB, respectively. The PRAM main memory has 4KB pages with a capacity of 512MB, and each page consists of 64 lines whose size is 64B.

In order to determine an appropriate DRAM buffer size, a preliminary experiment was performed with eight benchmarks from the SPEC CPU 2006, which are described in Table III. If the DRAM buffer is too small, the buffering effect is trivial. On the other hand, if the DRAM buffer is too large, all memory accesses are handled by the DRAM buffer, thereby masking the wear-leveling effect. As shown in Figure 9, the average number of write operations per page written to the PRAM main memory was evaluated. As the DRAM buffer size increases, the number of write operations decreases. Based on this result, we set the DRAM buffer size to 16MB because it is appropriate for most workloads for the SPEC CPU 2006.

The maximum write count and standard deviation of the write count were used as the main metrics. The maximum write count is important because it critically affects the PRAM lifetime. The standard deviation of write count represents how the page write counts are uniformly distributed. Using the metrics, the effects of an adaptive wear-leveling algorithm is given. The resulting PRAM lifetime is then discussed.

As comparisons schemes, the swapping and shifting proposed by Zhou et al. [2009] were chosen. The LRU scheme was used as the baseline buffering scheme because Zhou et al. [2009] did not use a DRAM buffer.

[1]http://www.spec.org.

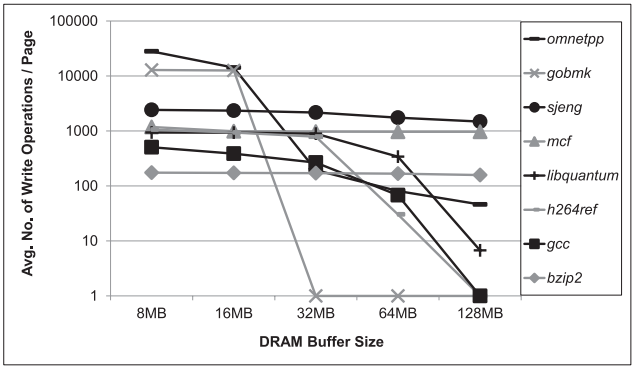
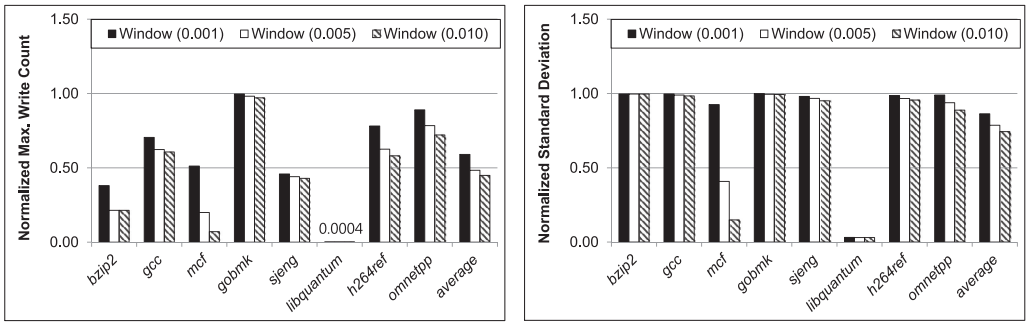


Fig. 9. Experimental results for setting DRAM buffer size.



(a) Maximum Write Count (b) Standard Deviation of Write Count

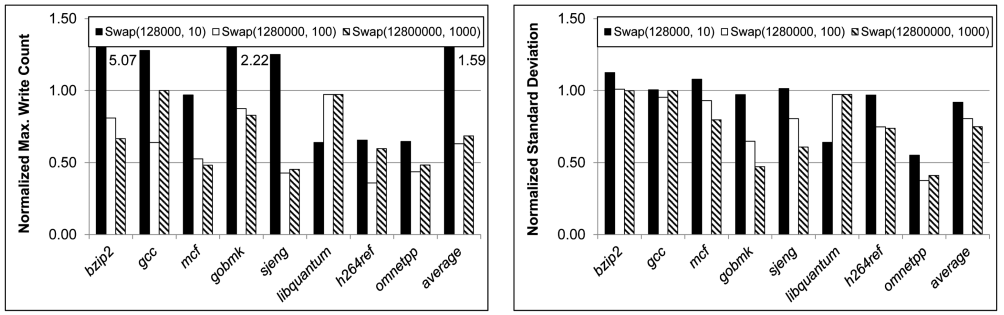
Fig. 10. PRAM-aware DRAM buffering effect with window sizes, (X) = (ratio for DRAM buffer size).

5.2. Experimental Result

In this section, the experimental results of the adaptive wear-leveling algorithm, which consists of a PRAM-aware DRAM buffering scheme, an adaptive multiple data swapping and shifting scheme, and a DRAM buffer-aware data swapping and shifting scheme are presented by showing the maximum write count and standard deviation of the write count. First, the effect of each scheme is evaluated according to various configurations. After evaluating each scheme's effects, the overall wear-leveling effects of the adaptive wear-leveling algorithm compared to the previous wear-leveling algorithm is evaluated.

5.2.1. PRAM-Aware DRAM Buffering Effects. In order to show the PRAM-aware DRAM buffering effects, we evaluated the maximum write count and standard deviation of the write count for three window sizes, 0.001, 0.005, and 0.010 of the DRAM buffer size [Ou et al. 2009]. The results were normalized to that of an LRU scheme.

As shown in Figure 10, the PRAM-aware DRAM buffering scheme, which considers the write count and clean data via an LRU scheme, can reduce the maximum write count from 3% in *gobmk* to 99.97% in *libquantum*, and the standard deviation of the write count from 0.3% in *bzip2* to 97% in *libquantum* compared to just an LRU scheme. The results are the most effective in *mcf* and *libquantum* because they are read-intensive workloads that make a lot of clean data. The clean data can cause the dirty data to be forcibly evicted in a LRU scheme, thus increasing the PRAM write count. On the other hand, the PRAM-aware DRAM buffering scheme can select the clean data



(a) Maximum write count (b) Standard deviation of write count

Fig. 11. Multiple data swapping effect as a function of operation timing and number of swapping pages, (X, Y) = (Total PRAM write count, number of swapping pages).

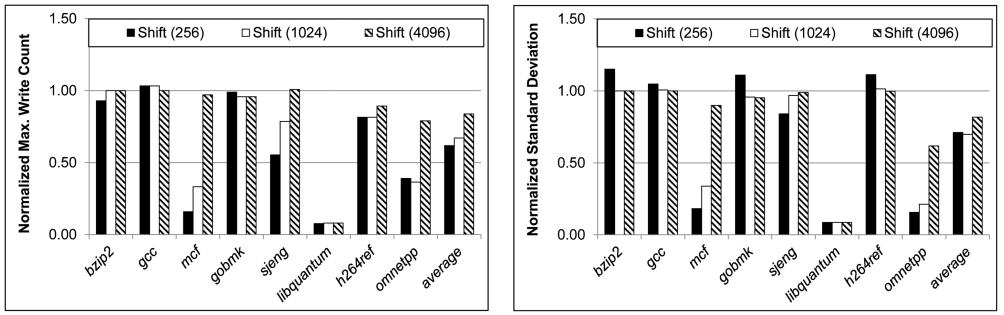
first in the window as a victim, thus making its effect significant in read-intensive workloads.

We can also show that the maximum write count is reduced as the window size increases. While a large window size reduces the maximum write count, the computation overhead for finding the victim in the window substantially increased. Therefore, a window size is set to 0.005 of the DRAM buffer size for further evaluation because its average maximum write count is similar to that of 0.01 (0.48 vs. 0.45), but its computation overhead is two times lower than that of 0.01.

From the experimental results, we concluded that the PRAM-aware DRAM buffering scheme can reduce the maximum write count and standard deviation of the write count as compared to the LRU scheme.

5.2.2. Effect of a Multiple Data Swapping and Line-Level Data Shifting Scheme. This section describes the effect of multiple data swapping and line-level data shifting schemes based on the operation timings and the number of swapping pages. We used an LRU scheme as the baseline buffering scheme only to show the effect of the swapping and shifting schemes. The results were normalized to that of an LRU scheme without any swapping and shifting scheme. For the multiple data swapping scheme, the operation timing was determined by the total PRAM write counts, which was set to 128000, 1280000, and 12800000. According to the operation timing, the number of swapping pages was set to 10, 100, and 1000. This is because an additional write overhead was maintained under 1% of the total PRAM write counts. For the line-level shifting scheme, the operation timing was determined by the PRAM write count per page, and it was set to 256, 1024, and 4096. From the evaluations with various configurations, we were able to find an appropriate configuration that shows the best wear-leveling effect.

Figure 11 shows the effect of the multiple data swapping scheme. The (X, Y) configuration represents the operation timing and the number of swapping pages. The operation timing is determined by the total PRAM write count. The multiple data swapping scheme can reduce both the maximum write count and standard deviation of the write count by evening out the write counts among the pages. The maximum write count is reduced by up to 64% in *h264ref*, and the standard deviation is reduced by up to 62% in *omnetpp*. In the (1280000, 100) configuration, four benchmarks, *gcc*, *sjeng*, *h264ref*, and *omnetpp*, show the best wear-leveling effects, while other benchmarks, *bzip2*, *mcf*, and *gobmk*, show the best wear-leveling effects in the (12800000, 1000) configuration. Only the *libquantum* benchmark achieved the best wear-leveling effect in the (128000, 10) configuration. For some workloads with the (128000, 10) configuration, the multiple data swapping scheme does not achieve the wear-leveling



(a) Maximum write count (b) Standard deviation of write count

Fig. 12. Line-level data shifting effect as a function of operation timing, (X) = (PRAM write count per page).

effect, but rather increases the maximum write count and standard deviation of the write count. This is because frequent swapping operations can make additional write counts to specific pages. Therefore, it is necessary to dynamically adapt parameters depending on the workload pattern.

Figure 12 shows the effect of the line-level data shifting scheme. The (X) configuration represents the operation timing, which is determined by the PRAM write count per page. The line-level data shifting scheme can also reduce the maximum write count and standard deviation of the write count by evening out the write counts among the lines of a page. The maximum write count is reduced by up to 92%, and the standard deviation of the write count is reduced by up to 91% in *libquantum*. Similar to the swapping operation, the configuration for the best wear-leveling effect is different according to the workload pattern, and the configurations of (256), (1024), and (4096) show an additional write overhead of up to 18.5%, 2.9%, and 0.5%, respectively. Therefore, it is also necessary to dynamically adapt the operation timing according to the workload pattern.

5.2.3. Adaptive Data Swapping and Shifting Effect. In order to show the adaptive data swapping and shifting effect, the increase in the maximum write count was evaluated by comparing a static data swapping and a shifting scheme. In both cases, an LRU scheme was used as the baseline buffering scheme. We used the (1280000, 100) configuration for the static data swapping and the (1024) configuration for the static data shifting scheme. For the adaptive data swapping and shifting scheme, the check period was set to 1280000 total PRAM write counts for the swapping operation, 256 PRAM write counts per page for the shifting operation, and a threshold value of 1 for the swapping operation and 7 for the shifting operation, which showed the best wear-leveling effect.

Figure 13 shows the increase in the maximum write count as the number of write operations increases. For the *bzip2*, *mcf*, *sjeng*, and *omnetpp* benchmarks, the adaptive data swapping and shifting scheme can reduce the maximum write count by 29%, 54%, 16%, and 34%, respectively, compared to a static data swapping and shifting scheme. This is due to the adaptive data swapping and shifting scheme being able to efficiently adapt the workload pattern. When the write access is skewed to a specific page, frequent swapping and shifting operations are performed, but no swapping and shifting operations are performed when the write access is uniform. On the other hand, the results between the adaptive scheme and static scheme are similar to the *gcc*, *gobmk*, *libquantum*, and *h264ref* benchmarks. This is because the workload pattern cannot be changed dynamically or the adaptive data swapping and shifting scheme predicts the changes in the workload pattern incorrectly. While the adaptive data swapping and shifting scheme is inefficient for some workloads, it is still necessary because the workload patterns changed differently with the PRAM-aware DRAM buffering scheme.

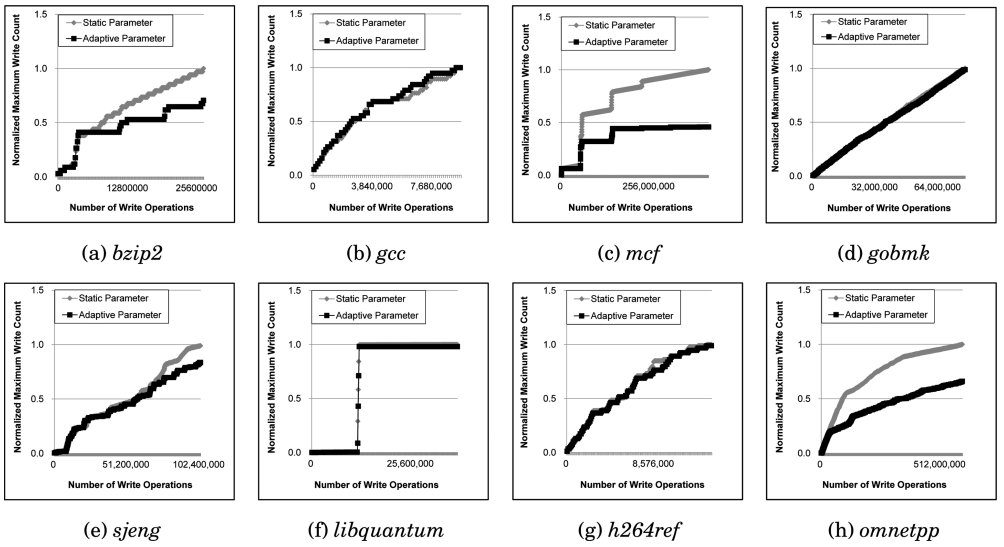


Fig. 13. Improvement of adaptive data swapping and shifting effects.

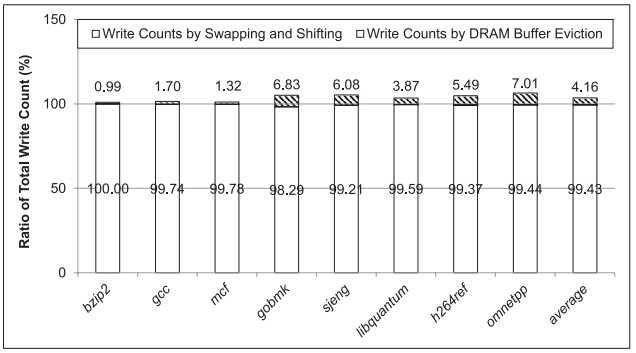
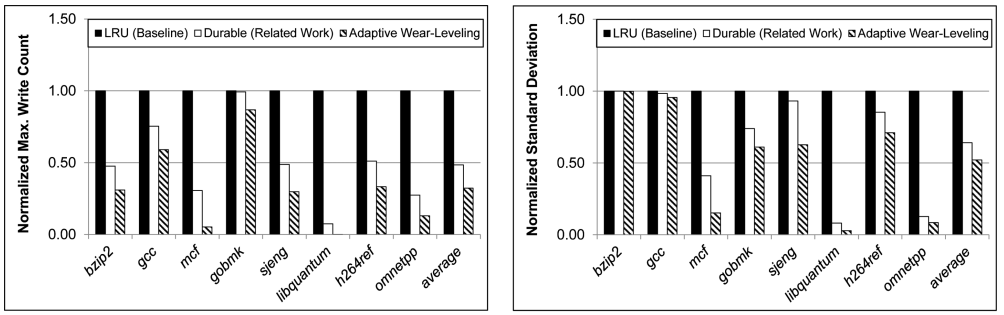


Fig. 14. Ratio of write counts to PRAM main memory.

5.2.4. Effect of DRAM Buffer-Aware Swapping and Shifting Scheme. In order to show the DRAM buffer-aware swapping and shifting effect, the ratio of total write counts that consists of write counts incurred by the DRAM buffer eviction and the swapping and shifting operation is given. For this experiment, the (1280000, 100) configuration and (1024) configuration were set for the swapping and shifting, respectively.

Figure 14 shows the ratio of PRAM write counts. The reason for the write count portion of the DRAM buffer eviction being less than 100% is that the DRAM buffer-aware scheme can reduce the write count by marking dirty data as clean during the swapping and shifting operations. In *gobmk*, the total write count is reduced by up to 1.71%. By reducing the PRAM write counts with the DRAM buffer-aware swapping and shifting scheme, the wear-leveling effect can be improved.

5.2.5. Overall Wear-Leveling Effect Compared with Previous Algorithm. We evaluated the overall wear-leveling effect by comparing the adaptive wear-leveling algorithm with the previous swapping and shifting proposed by Zhou et al. [2009] based on an LRU scheme. First, the maximum write count and standard deviation of the write count were evaluated. Then, the PRAM lifetime was calculated.



(a) Maximum Write Count (b) Standard Deviation of Write Count

Fig. 15. Effect of adaptive wear-leveling technique compared to previous wear-leveling algorithm.

For the adaptive wear-leveling algorithm, we set the window size to 0.005 of the DRAM buffer size, the check period to 1280000 total PRAM write counts for the swapping and 256 PRAM write counts per page for the shifting, the threshold value to 1 for the swapping and 7 for the shifting. For the previous wear-leveling algorithm, we set the swapping period to 1280000 total PRAM write counts, a shifting period to 1024 PRAM write counts per page, and a page size to 256*4KB.

Figure 15 shows the effect of the adaptive wear-leveling algorithm compared to the previous wear-leveling algorithm. The results are normalized to that of an LRU scheme without any swapping and shifting schemes. While the previous wear-leveling algorithm can reduce the maximum write count and the standard deviation of write count compared to only the LRU scheme, the adaptive wear-leveling algorithm achieves a greater wear-leveling effect than the previous wear-leveling algorithm. As compared to the previous wear-leveling algorithm, the adaptive wear-leveling algorithm can reduce the maximum write count from 12.6% in the *gobmk* benchmark to 99.4% in the *libquantum* benchmark, and the standard deviation of write count from 0.2% in the *bzip2* benchmark to 64.0% in the *libquantum* benchmark.

5.2.6. Lifetime Improvement. By using the results from the previous section, the PRAM lifetime based on Equation (2) was calculated. This function is a modified version of a previous system lifetime provided by Qureshi et al. [2009b]. The reason why we modified the lifetime is that they assumed that the write counts are distributed uniformly across the entire main memory system.

$$System\ Lifetime\ (sec) = \frac{E_{max} \cdot S}{B_{eff} \cdot W_{max}}$$

, where

E_{max}: Endurance of PRAM Main Memory,
S: Size of PRAM Main Memory (GB),
B_{eff}: Effective Write Traffic (GBps),
W_{max}: Effective Maximum Write Count. (2)

We used E_{max} as the endurance of the PRAM main memory, and S as the size of the PRAM main memory. In this experiment, we set E_{max} to 10^8 [Zhou et al. 2009; Lee et al. 2009; Zhang and Li 2009; Kong and Zhou 2010] and S to 512MB. For DDR3 DRAM, the maximum bandwidth is 10GB/s in both the read and write operations, but the PRAM maximum read bandwidth is 2 times lower and the maximum write bandwidth is 5 times lower than that of DRAM. Therefore, the PRAM maximum read bandwidth

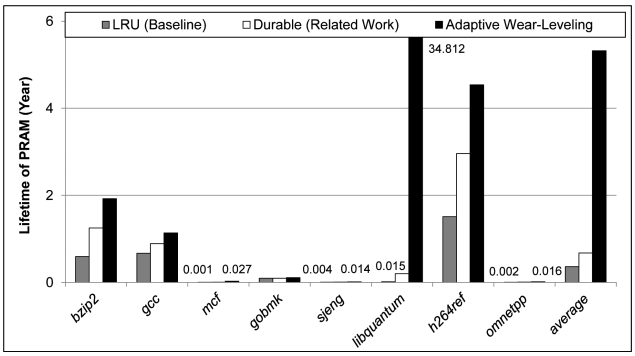


Fig. 16. Lifetime of PRAM main memory.

of 5GB/s and the maximum write bandwidth of 2GB/s were used. From these values, the effective write traffic (B_{eff}) was used instead of the memory access traffic, because only writes to the PRAM main memory affect the PRAM lifetime. B_{eff} was calculated using ((2 · Write_Ratio+5 · Read_Ratio) · PRAM_Write_Ratio). PRAM_Write_Ratio was used because a DRAM buffer was used, thereby considering the ratio of actual PRAM write operations to total memory operations for calculating the effective write traffic. Actually, because of other factors (the victim selection and the idle time between the memory requests), we cannot achieve the maximum bandwidth for all memory accesses. However, in order to show the worst-case lifetime, we assume that our approach can be performed with the maximum bandwidth. W_{max} was calculated by (max_write_count · $\frac{coverage(MB)}{512(MB)}$). This is because each benchmark has its own memory coverage, which means a range of virtual memory addresses, and it is performed within the coverage in our evaluation. Therefore, we normalized the maximum write count by using the PRAM main memory size and the memory coverage of benchmarks. With these parameters, the system lifetime was finally calculated.

Figure 16 shows the lifetime of the PRAM main memory. For the *libquantum* benchmark, the PRAM lifetime increased up to 34.81 years compared to 0.2 years of the previous wear-leveling algorithm, due to the prevention of skewed write operations to a specific page by considering the clean data and write count in the DRAM buffer. For the *bzip2*, *gcc*, and *h264ref* benchmarks, the PRAM lifetime increased from 1.25, 0.89, and 2.96 years of the previous wear-leveling algorithm to 1.88, 1.08, and 4.54 years, respectively. For the *mcf*, *sjeng*, and *omnetpp* benchmarks, the PRAM lifetime of the baseline system is only 0.001, 0.004, and 0.002 years, respectively. This is because they have large maximum write counts compared to other benchmarks. Nevertheless, 18.8 times, 3.36 times, and 7.7 times longer PRAM lifetime was achieved compared to the baseline system. For the *gobmk* benchmark, only 1.15 times longer PRAM lifetime was achieved compared to an LRU only scheme because the *gobmk* benchmark has a uniform write pattern. On average, the PRAM lifetime increased from 0.36 years with the baseline and 0.68 years with the previous wear-leveling algorithm to 5.32 years with the proposed algorithm. Except for the *libquantum* benchmark, which had an extreme lifetime increase, the PRAM lifetime increased to 1.11 years.

5.2.7. Additional Write Overhead by Swapping and Shifting Operations. Although the swapping and shifting operations have improved the wear-leveling effect, they make additional read and write operations. As shown in Figure 17, we evaluated the ratio of the additional write count to the total PRAM write count. For the previous wear-leveling algorithm, the ratio was 4.9% on average (from 1.1% in the *bzip2* benchmark to 8.8%

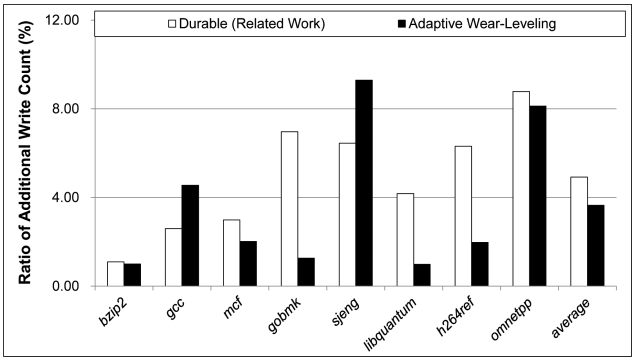


Fig. 17. Ratio of additional write count to total PRAM write count.

in the *omnetpp* benchmark). For the adaptive wear-leveling algorithm, the ratio was 3.7% on average (from 0.98% for the *libquantum* benchmark to 9.3% in the *sjeng* benchmark). Compared to the previous wear-leveling algorithm, the adaptive wear-leveling algorithm shows a lower overhead on average with a higher wear-leveling effect. This is because the adaptive wear-leveling algorithm can predict the future workload pattern. Thus, no operation is performed when the write accesses are evenly distributed. The DRAM buffer-aware swapping and shifting scheme also reduces unnecessary writes. In case of the *gcc* and *sjeng* benchmarks, frequent shifting operations are necessary due to skewed write accesses to a specific line. Although the adaptive wear-leveling algorithm shows a higher ratio of additional operations in those benchmarks, it shows a higher wear-leveling effect than the previous wear-leveling algorithm.

6. CONCLUSION AND FUTURE WORK

An adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer was proposed. This adaptive wear-leveling algorithm improves the PRAM lifetime via a PRAM-aware DRAM buffering scheme, an adaptive multiple data swapping and shifting scheme, and a DRAM buffer-aware data swapping and shifting scheme. By considering the write count and clean data based on an LRU scheme in the DRAM buffer, the PRAM write count is reduced and a skewed write count of specific pages is prevented. Via the adaptive multiple data swapping and shifting scheme, the wear-leveling effect of the previous static swapping and shifting scheme can be improved by adapting the workload pattern. Finally, the DRAM buffer-aware data swapping and shifting scheme reduces the overhead of additional swapping and shifting operations by reducing unnecessary write counts. With the adaptive wear-leveling algorithm, the maximum write count was reduced by up to 64.0%, and the standard deviation of the write count by up to 99.4% compared to the previous wear-leveling scheme. Therefore, the PRAM lifetime also increased from 0.68 years with the previous wear-leveling algorithm to 5.32 years with the adaptive wear-leveling algorithm with a 3.6% additional write overhead, on average.

While the adaptive wear-leveling algorithm improves the PRAM lifetime, there are several areas that could be further improved. In future work, the reduction of storage and performance overhead for the adaptive wear-leveling algorithm will be the main focus. The storage overhead for write counts and LRU bookkeeping overhead can be reduced by using bit arrays and a clock algorithm, respectively. The performance overhead incurred by the adaptive wear-leveling algorithm can be reduced by exploiting the write pausing and cancellation scheme in PRAM [Qureshi et al. 2010] or the bank usage table scheme with the multibank characteristics [Delaluz et al. 2002]. We can

also overcome a trade-off between the wear-leveling effect and the performance overhead by implementing a cycle-accurate memory controller. Finally, we will consider a multiprogrammed environment as future work. Because the multiprogrammed environment makes more dynamic workload patterns, we anticipate that our adaptive wear-leveling algorithm will improve the PRAM endurance compared to previous static wear-leveling algorithms as well.

REFERENCES

Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo. 2010. Improving flash wear-leveling by proactively moving static data. *IEEE Trans. Comput.* 59, 1, 53–65.

S. Cho and H. Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO 42*). 347–357.

V. Delaluz, A. Sivasubramaniam, M. T. Kandemir, N. Vijaykrishnan, and M. J. Irwin. 2002. Scheduler-based DRAM energy management. In *Proceedings of the Design Automation Conference (DAC)*. 697–702.

G. Dhiman, R. Ayoub, and T. Rosing. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Annual Design Automation Conference* (*DAC*). 664–469.

J. Dong, L. Zhang, Y. Han, Y. Wang, and X. Li. 2011. Wear rate leveling: Lifetime enhancement of PRAM with endurance variation. In *Proceedings of the Design Automation Conference (DAC)*. 972–977.

A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé. 2010. Increasing PCM main memory lifetime. In *Proceedings of the Conference on Design, Automation and Test in Europe* (*DATE*). 914–919.

J. Guo, J. Yang, Y. Zhang, and Y. Chen. 2013. Low cost power failure protection for MLC NAND flash storage systems with PRAM/DRAM hybrid buffer. In *Proceedings of the Conference on Design, Automation and Test in Europe* (*DATE*). 859–864.

E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda. 2010. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *Proceedings of the 15th Edition of ASPLOS on Architectural Support For Programming Languages and Operating Systems* (*ASPLOS*).

H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee. 2006. FAB: Flash-aware buffer management policy for portable media players. *IEEE Trans. Consum. Electron.* 52, 2, 485–493.

J. Kong and H. Zhou. 2010. Improving privacy and lifetime of PCM-based main memory. In *Proceedings of IEEE DSN*. 333–342.

B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. 2009. Architecting phase change memory as a scalable DRAM alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (*ISCA*). 2–13.

S. Lee, D. Shin, and J. Kim. 2008. Buffer-aware garbage collection for NAND flash memory-based storage systems. In *Proceedings of the International Workshop on Software Support for Portable Storage* (*IWSSPS*).

C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. 2005. PIN: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI*). 190–200.

Y. Ou, T. Härder, and P. Jin. 2009. CFDC: A flash-aware replacement policy for database buffer management. In *Proceedings of the Fifth International Workshop on Data Management on New Hardware* (*DaMoN*). 15–20.

H. Park, S. Yoo, and S. Lee. 2011a. Power management of hybrid DRAM/PRAM-based main memory. In *Proceedings of the 48th Design Automation Conference* (*DAC*). 59–64.

S. K. Park, H. Seok, D.-J. Shin, and K. H. Park. 2012. PRAM wear-leveling algorithm for hybrid main memory based on data buffering, swapping, and shifting. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (*SAC*). 1643–1644.

S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Ki M, and J. Lee. 2006. CFLRU: A replacement algorithm for flash memory. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (*CASES*). 234–241.

Y. Park, S.-H. Lim, C. Lee, and K. H. Park. 2008. PFFS: A scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash. In *Proceedings of the ACM Symposium on Applied Computing* (*SAC*). 1498–1503.

Y. Park, S. K. Park, and K. H. Park. 2010. Linux kernel support to exploit phase change memory. In *Proceedings of the 12th Annual Linux Symposium* (*LS*).

Y. Park, D.-J. Shin, S. K. Park, and K. H. Park. 2011b. Power-aware memory management for hybrid main memory. In *Proceedings of the 2nd International Conference on Next Generation Information Technology* (*ICNIT*). 82–85.

M. K. Qureshi, M. Franceschini, and L. A. Lastras-Montano. 2010. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of IEEE HPCA*. 1–11.

M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. 2009a. 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* (*MICRO*). 14–23.

M. K. Qureshi, V. Srinivasan, and J. A. Rivers. 2009b. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (*ISCA*). 24–33.

L. E. Ramos, E. Gorbatov, and R. Bianchini. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing* (*ICS*). 85–95.

D. Roberts, T. Kgil, and T. Mudge. 2009. Using non-volatile memory to save energy in servers. In *Proceedings of the Conference on Design, Automation and Test in Europe* (*DATE*). 743–748.

A. Seznec. 2010. A phase change memory as a secure main memory. *Comput. Arch. Lett.* 9, 1, 5–8.

B. Wongchaowart, M. Iskander, and S. Cho. 2010. A content-aware block placement algorithm for reducing PRAM storage bit writes. In *Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies* (*MSST*). 1–11.

B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu. 2007. A low power phase-change random access memory using a data-comparison write scheme. In *Proceedings of IEEE ISCAS*. 3014–3017.

N. Ye, Q. Chen, and C. M. Borror. 2004. EWMA forecast of normal system activity for computer intrusion detection. *IEEE Trans. Rel.* 53, 4, 557–566.

W. Zhang and T. Li. 2009. 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* (*MICRO 42*). 2–13.

P. Zhou, B. Zhao, J. Yang, and Y. Zhang. 2009. A durable and energy effcient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (*ISCA*). 14–23.

Received December 2012; revised September 2013, November 2013; accepted November 2013