Bloom Filter-based Dynamic Wear Leveling for Phase-Change RAM

Joosung Yun, Sunggu Lee, Sungjoo Yoo
Department of Electronic and Electrical Engineering
Pohang University of Science and Technology (POSTECH)
{acajuri, slee, sungjoo.yoo}@postech.ac.kr

ABSTRACT

Phase Change RAM (PCM) is a promising candidate of emerging memory technology to complement or replace existing DRAM and NAND Flash memory. A key drawback of PCMs is limited write endurance. To address this problem, several static wear-leveling methods that change logical to physical address mapping periodically have been proposed. Although these methods have low space overhead, they suffer from unnecessary data migrations thereby failing to exploit the full lifetime potential of PCMs. This paper proposes a new dynamic wear-leveling method that reduces unnecessary data migrations by adopting a hot/cold swapping-based dynamic method. Compared with the conventional hot/cold swapping-based dynamic method, the proposed method requires only a small amount of space overhead by applying Bloom filters to the identification of hot and cold data. We simulate our method using SPEC2000 benchmark traces and compare with previous methods. Simulation results show that the proposed method reduces unnecessary data migrations by 58~92% and extends the memory lifetime by 2.18~2.30 times over previous methods with a negligible area overhead of 0.3%.

1. Introduction

Resistive memory is a new type of memory being developed in many laboratories. Unlike most previous capacitive memory devices that store data using electrical charge in capacitors, resistive memories store data using differences in resistivity brought about by material characteristics. Because of this feature, resistive memory is nonvolatile. The resistive memory has often better scaling capability than the DRAM since it does not suffer from the difficulty of keeping electron charge in a smaller capacitor as in the DRAM. Phase Change Memory (PCM) is a type of resistive memory that uses Ge, Sb, Te and other materials to produce phase changes in atomic structures resulting in changes in resistivity [1]. Many studies [2, 3] have shown that PCM can give benefits when utilized as an additional memory structure in the memory hierarchy and storage.

Like Flash memory, a key drawback of PCMs is that there is a limit to the number of times that a memory cell can be written. A PCM cell can be written 10^7-10^8 times before it wears out and fails [1,4]. Although the endurance of a PCM cell is higher than a Flash memory cell, wear-leveling technology, which evenly distributes writes across cells, is still needed for PCMs to be used as a part of the main memory or as a write buffer in the storage. Unlike Flash memory, which must erase an entire block before rewriting that block with new data, memory cells in PCMs can be modified one cell at a time. This leads to fine-grained methods for wear leveling.

Conventional wear leveling methods can be classified into two types, dynamic and static methods, depending on whether the hotness of data is utilized or not. In the dynamic methods, hot data (frequently written data) are identified by counting the number of writes and their physical locations are changed (with the physical locations of cold data) when the write counts exceed the given threshold. In the static method, the mapping from logical to physical addresses is changed periodically without considering hot/cold data locations. The dynamic methods give better wear leveling with longer lifetime and smaller data migration overhead. However, they suffer from high area cost to maintain the write count information.

In this paper, we propose a low cost solution by applying the Bloom filter [5] to the management of write counts. The basic idea is to exploit the fact that writes have spatial localities in the entire address space. Thus, some regions of address space receive frequent writes while the other regions have few write requests. In such a case, as will be explained later, the Bloom filter is useful to identify hot data addresses with a very small area overhead. We simulate our method and previous methods [6, 7] and compare lifetime and overhead.

The remainder of this paper is organized as follows. Section 2 reviews previous work. Section 3 gives the basic idea. Section 4 explains the proposed method. Section 5 reports experimental results. Section 6 concludes the paper.

2. Related Work

There have been presented several studies on static wear leveling for PCM. In [8], two rotation-based methods are proposed, fine-grain one called row shifting & coarse-grain one called segment sifting. Row shifting rotates the start address of row by one byte at a time for each shift interval based on the number of write count per row. Segment shifting is a coarse-grained rotation method using the number of write count per segment (i.e., multiple rows) instead of a single row. In [6], the *start-gap* method first randomizes the mapping from logical to physical addresses to distribute write traffics with high spatial locality (e.g., frequently accessing neighbor addresses). Then, by combining a rotation-based wear leveling utilizing start and gap registers, logical to physical address mapping is changed. In [7], *security-refresh* performs a two-level randomization for logical to physical address mapping to avoid PCM wear out due to malicious attacks, e.g., repeat address attacks.

The static methods have the advantage of small space overhead for the write count information. However, as will be shown in our experiments, they suffer from unnecessary data swapping thereby failing to exploit the full lifetime potential of PCMs.

Dynamic wear leveling can give longer lifetime than static one. Dynamic wear leveling has been actively studied for Flash memories [9, 10, 11, 12]. These methods try to achieve uniform writes by swapping addresses using write count information at the granularity of erase operation, i.e., block (e.g., 256KB=64*4KB). Since memory cells in PCMs can be modified one cell at a time, a simplistic adoption of dynamic wear leveling methods from Flash memories will give prohibitively large area overhead since it would require write information *per cell*.

In [13], a method called *wear rate leveling* is proposed where the write count information is managed at the granularity of data

978-3-9810801-8-6/DATE12/©2012 EDAA

regions. Periodically, data regions are sorted based on their write counts. Then, the most frequently written data regions are mapped onto strong (in terms of process variation) PCM cells thereby improving PCM lifetime. Like existing dynamic methods for Flash memories, this method has also the limitations of large area overhead to maintain the write count information. In addition, the timing-consuming sorting operation is required during runtime. In our method, we reduce the area overhead of write count information by utilizing the Bloom filter and the runtime overhead of sorting to find the hot addresses by utilizing an efficient hot-cold list management.

3. Preliminary and Basic Idea

3.1 Bloom Filter

The Bloom filter is a space-efficient probabilistic data structure [5]. It is used to query whether an element is in a data set or not. The original Bloom filter is a bit array of m bits. In the initial empty state, all entries in this bit array are set to 0. Then, we define k different hash functions, each of which maps a data element to one of m bit array positions. When a new data element is added to the data set, all k hash functions are applied to this data element, and each corresponding entry in the bit array is set to 1. After a large number of data elements have been added to the database in this manner, a large number of entries in the bit array will be 1; however, assuming that the bit array is sufficiently large (m is large), the bit array will still have more 0 entries than 1 entries. To query for the existence of a data element in the data set, the user simply needs to check the k positions which the k hash functions, when applied to this data element, point to. If any of these bits is 0, then this data element is not in the data set. If all bits are 1, then there is a positive probability that this data element is in the data set.

Because the Bloom filter (often called binary Bloom filter) uses this type of bit array, counting the same data elements is not possible. Thus, in our work, we utilize a counting Bloom filter that consists of a counter array. Thus, whenever a new entry is added, the corresponding counter(s) is incremented.

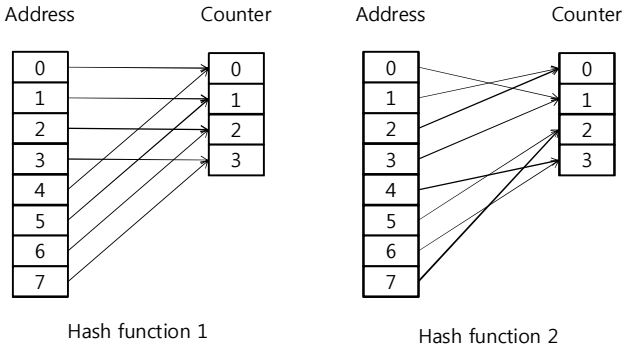

Fig. 1. Hash functions mapping addresses to Bloom filter counter arrays.

3.2 Bloom Filter-based Hot Data Identification

Let us consider a specific example of Bloom filter as shown in Fig 1. There are 8 addresses and 4 counters. Note that we reduce the number of counters from 8 to 4 by utilizing the Bloom filter while the conventional dynamic wear leveling has 8 counters in this case. Fig. 1 shows the two hash functions, i.e. two mapping relationships between addresses and counters. Fig. 2 shows two example scenarios. In Fig. 2 (a), suppose that all addresses are written 5 times. Then, all counters have the same value of 20 as shown in Fig. 2 (a). In the second scenario of Fig. 2 (b), suppose that all addresses are written 5 times except for address 0, which is written 50 times. Then, the values of counters 0 and 1 are both 65, while the others are 10. As shown in Fig. 2 (b), in the case that there are hot addresses, the Bloom filter can identify them since the corresponding counters give higher values than the other counters.

Note again that the Bloom filter-based hot address identification requires less area overhead, i.e., less counters, for write count information.

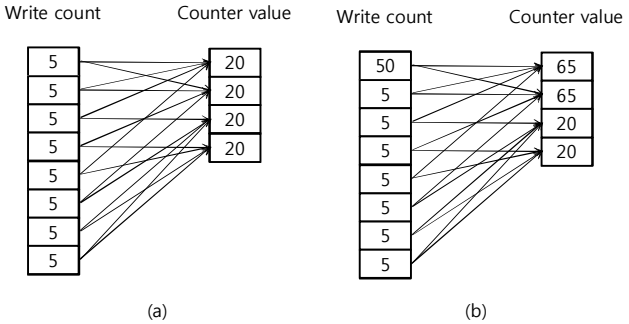

Fig. 2. Two example scenarios (uniform writes and one hot address).

Fig. 3 illustrates in more detail how to identify hot addresses utilizing the counting Bloom filter. To determine whether an address is hot or not, whenever a data is written, the corresponding entries in the Bloom filter counter array are checked. In Fig. 3, if both of the two counters give higher values than a threshold (e.g., 30 in the figure), the address is classified as a hot address. In such a case, we perform wear leveling, i.e., swap the physical locations between the hot address and one of cold addresses. Then, we manage the information of swapping (i.e., original address and swapped address) in a hot-cold list L.

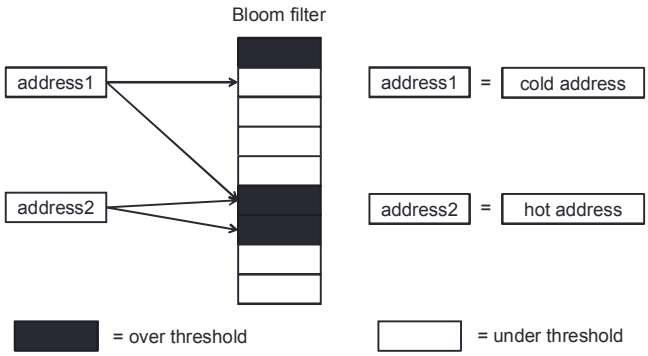

Fig. 3. Identifying hot addresses.

The counting Bloom filter can give false positives and false negatives. A false (negative) positive is a prediction error that the Bloom filter predicts existence (non-existence) in case of non-existence (existence). For instance, in Fig. 2 (b), the second address has 5 writes. However, the Bloom filter can identify it as a hot address since the values of its Bloom filter counters both exceed the threshold. Note that such errors occur in proportion to the write

probability of the second address. Even though the write probability of cold addresses is typically very low, the adverse effects of such errors need to be evaluated.

4. Bloom Filter-based Wear Leveling

4.1 Solution Overview

Fig. 4 shows an overall picture of the proposed method. When we wish to access (read from or write to) a specific address, we first check the hot-cold list *L* (step 1 in Fig.4). If the address is not found in *L*, we access its original location (step 2). If the *L* includes that address, we access the swapped address, i.e., the address directed to by the hot-cold list (step 3). Whenever data is written to a specific address, the Bloom filter (having two hash functions in our work) is updated, i.e., the counters that are indicated by the two hash functions for that address are incremented (step 4).

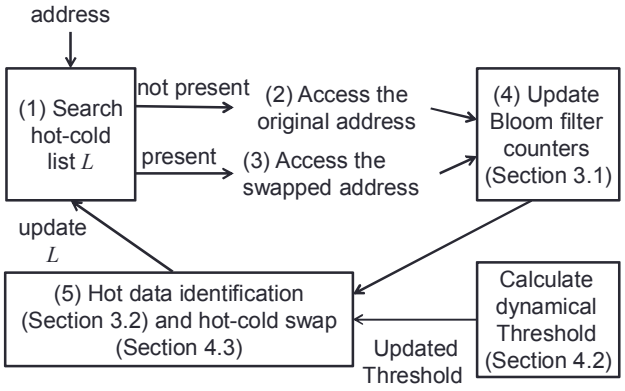

Fig. 4. Overview of Bloom filter-based wear-leveling.

In case of writes, after updating the Bloom filter, we check to see if a hot/cold swapping is needed. As mentioned in Section 3, if all the counters of written data exceed the threshold, then the address of write data is identified as a hot address, and is swapped with a randomly selected address (step 5 in Fig. 4).

The hotness of data varies during runtime. In our work, we propose a novel idea to improve the performance of Bloom filter-based hot address identification by adapting the threshold to dynamically changing write behavior (Section 4.2). In addition, in order to minimize the adverse effects of false positive (i.e., erroneously identifying a cold address as a hot one), we present a method of hot-cold list management to filter out such addresses from the hot-cold list (Section 4.3).

4.2 Dynamically Changing Threshold

In this subsection, we propose two policies to change the threshold during runtime by adapting to the dynamically changing write behavior. Given a program, we perform a design-time evaluation on which policy is better for the given application and then, during runtime, we apply the best policy to the program. During runtime, we dynamically adjust the threshold according to the policy only when there is a significant change in the statistics of Bloom filter counter values. In our implementation, we applied the policy when the standard deviation of Bloom filter counter values is under 10% or over 90% of its maximum.

Policy 1. In the case that the write count behavior exhibits a near even distribution, if the standard deviation of Bloom filter counter values is increasing, the threshold value should increase. If the standard deviation is decreasing, the threshold value should decrease.

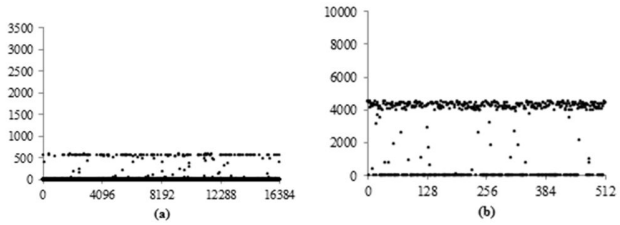

Fig. 5. A near even write count distribution (a) and a corresponding distribution of Bloom filter counter values (b)

Policy 1 is effective when the write behavior of the program exhibits a near even distribution, i.e., the absolute amount of write count difference between hot and cold addresses is small. In such a case of originally even write distribution, we need to try to avoid unnecessary hot/cold swapping since hot/cold swapping will increase PCM writes thereby reducing lifetime without contributing to the even-ness of write counts. Thus, if the write behavior changes dynamically while still keeping the even-ness, we need to adjust the threshold while trying to minimize the swapping overhead. In such a case, as stated in Policy 1, even though the standard deviation of Bloom filter counters increases, if the even-ness of write distribution is still maintained, then we try to reduce hot/cold swapping by increasing the threshold.

Fig. 5 exemplifies an original near even write count distribution (Fig. 5 (a)) and a corresponding distribution of Bloom filter counter values obtained from trace 183.equake in SPEC2000. The x-axes in Fig. 5 (a) and (b) represent the counter of original address and the Bloom filter counter, respectively. The y-axes are write counts. Note that the distribution of Bloom filter counter values is correlated with that of the original write distribution since each Bloom filter counter represents the sum of write counts in a sample set of the original write counters set. If the statistics of Bloom filter counter values is like the one in Fig. 5 (b), we can consider the original write distribution will be near even. Thus, we need to try to minimize hot/cold swapping by adjusting the threshold according to Policy 1.

Policy 2. In the case that there are hot addresses, if the standard deviation of Bloom filter counter values is increasing, the threshold value should decrease. If the standard deviation is decreasing, the threshold value should increase.

Policy 2 is for programs having hot addresses. In such a case, it is important to increase the sensitivity of hot address identification in order to maximize the effects of hot/cold swapping. Fig. 6 illustrates such a case obtained from the same program as in Fig. 5. Fig. 6 (a) shows that there are hot addresses which have large number of writes (e.g., more than 200) while cold addresses each have only 0~199 writes. In this case, if the distribution of Bloom filter counter values becomes wider, then there will be more and/or stronger hot addresses. Thus, it is required to identify hot addresses as soon as possible and to perform hot/cold swapping. To do that, as stated in Policy 2, in such a case, if the standard deviation of Bloom filter counter values increases, we decrease the threshold for a faster identification of hot addresses.

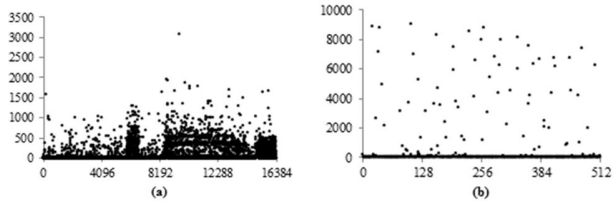

Fig. 6. Write count distribution with hot addresses (a) and a corresponding distribution of Bloom filter counter values (b)

Note that we find a suitable policy for each program during design time and then apply it during runtime. In each policy, we periodically calculate the standard deviation of Bloom filter counter values and change the threshold value appropriately.

4.3 Three-Tier Hot-cold List Management

Due to the false positive problem, non-hot addresses can be judged to be hot ones. In such a case, the newly identified hot address (in reality, a non-hot address) evicts from the hot-cold list one of existing hot addresses, which can adversely affect the effectiveness of the proposed method. To address this problem, we suggest a three-tier hot-cold list management. Our idea is (1) to double-check newly identified hot addresses and (2) to keep in the hot-cold list as many real hot addresses as possible. To do that, we divide the entire hot-cold list into three smaller lists as shown in Fig. 7. Lists 1 and 2 contain strong and weak hot addresses, respectively. List 3 has new hot addresses selected in step 5 of Fig. 4.

When a new address is included into the hot-cold list, it is first inserted into list 3 whose replacement policy is First-In-First-Out (FIFO). If an address in list 3 is accessed, then it is promoted to the last position in list 2 (denoted with 'Entry point' in Fig. 7). If an address in lists 1 and 2 is accessed, it is promoted by one position towards the top position of each list. Periodically, the lower part (20%) of list 1 and the upper part of group 2 are shuffled as shown in Fig. 7.

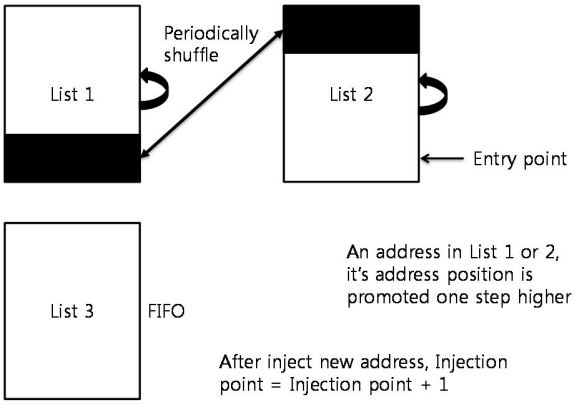

Fig. 7. Three-tier hot-cold list

The three-tier list structure can filter out non-hot addresses which are selected due to the false positive problem since non-hot addresses have little probability of re-reference (for it to promote to list 2) until it is evicted from list 3 by the FIFO policy. In this three-tier structure, we perform hot/cold swapping when an address is inserted to list 3.

5. Experimental Results

5.1 Simulation Environment

To simulate our method, we extracted memory access traces based on parameters used in actual working systems, as shown in Table 1. The event-driven multicore simulator McSim [14] was used to obtain write eviction traces from the L1 cache. These traces were then used as the write address input to our simulation model of PCM-based memory subsystem. Table 2 shows the simulation setting. Because cache lines are 64B, we set our address granularity of hot/cold swapping to be 64B.

Table 1 System configuration for simulation

Component	Details
CPU core	In-order x86 core, 450 MHz
L1 cache	4-way, 16KB/32KB, I/D cache, 64B cache line, 1 cycle, 450MHz
PRAM	32b DDR2 interface @ 400MHz, tCL/tRP/tRCD=6/72/25 cycles.

Table 2 Simulation setting

Figure	Size
Granularity	64Byte
PRAM	1G Byte
2-level Bloom filter	256 x 20 Bit
1-level Bloom filter	256 x 13 Bit
Groups	4096 addresses per group
Hot-cold List	256 x 12 Bit

A management of a single very large Bloom filter is not efficient, e.g., in terms of power consumption. In our implementation, we built a two-level Bloom filter as shown in Fig. 8. It partitions the address space into smaller groups and manages per-group Bloom filters and a global Bloom filter. In order to avoid the overflow of Bloom filter counters, the counter values are divided by 2 periodically (per 20000 writes).

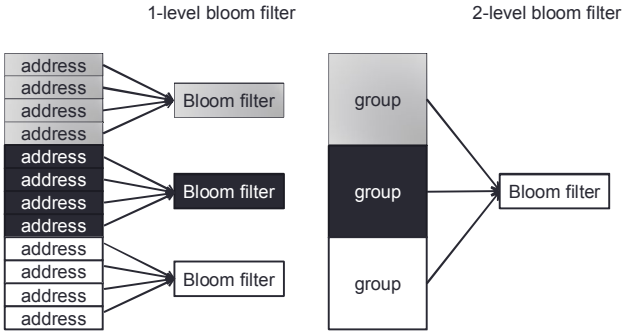

Fig. 8. 2-level Bloom filter.

Based on the simulation settings shown in Table 2, we can calculate the required space overhead of the proposed method as follows.

$$\text{Space overhead} = \text{Group} \times (\text{1-level Bloom filters+hot-cold list}) + (\text{2-level Bloom filter+hot-cold list})$$
$$= 4096 \times (256 \times 13 + 256 \times 12) + 256 \times 20 + 256 \times 12$$
$$= 3201\text{KByte} = 3.16\text{MByte}$$

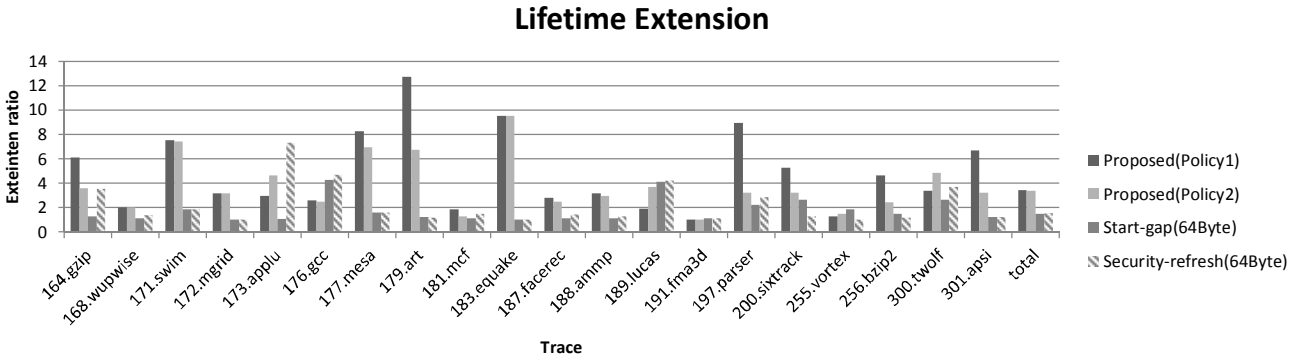

Fig. 9. Lifetime extention compared with previous methods

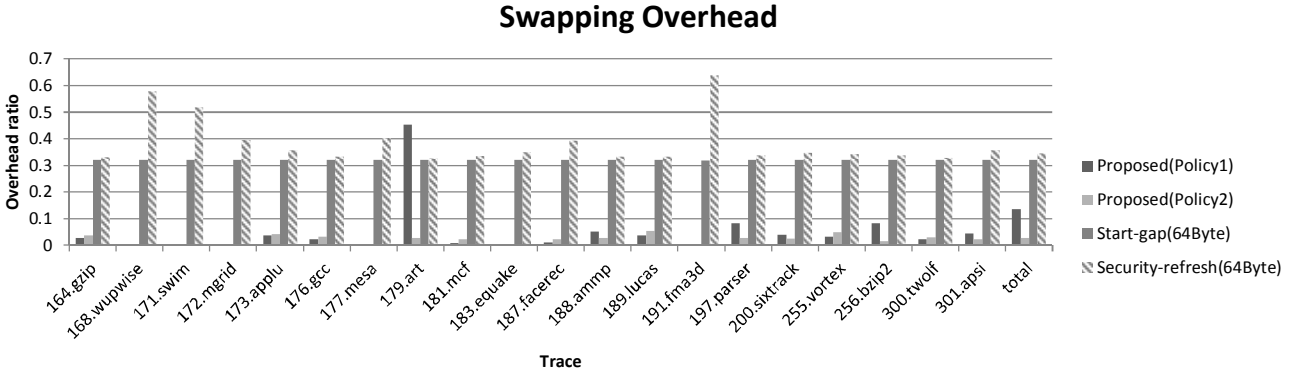

Fig. 10. Swapping overhead compared with previous methods

The space overhead is 0.31% of total memory size (1GB) which is significantly smaller than that of an ideal case of hot/cold swapping (11.4%, 12b per 64B) where each data of swapping granularity (in our case, 64B) has its own counter. Note that the sorting overhead with such a large number of counters is prohibitively large in the ideal case. For the implementation of our method, we assume storing the Bloom filters and hot-cold lists in PCM and utilizing a caching mechanism similar to TLB (translation look-aside buffer) used for virtual-to-physical address translation.

5.2 Comparison with Previous Methods

The proposed method was compared with two well-known previous methods: start-gap [5] and security-refresh [6]. We simulated 20 SPEC2000 traces. Fig. 9 shows a comparison of lifetime extension ratios, and Fig. 10 shows a comparison of swapping overhead. Fig. 11 is a summary of Figs. 9 and 10. It also includes the ideal case. To obtain the ideal case, we first sorted per-address (64B) write counts in the traces. For a given target lifetime (i.e., maximum write count), we selected those data whose write counts exceed the maximum write count. Then, we calculated the required number of swappings in order to meet the write counts of physical locations for those data under the maximum write count. In Fig. 11, the distance from the origin represents the quality of wear leveling method. Thus, the farther a solution is located from the origin, the better quality the solution gives.

The simulation results show that the proposed method reduces unnecessary data swapping by 58~92% and extends the memory lifetime by 2.18~2.30 times over previous methods with a negligible area overhead of 0.3%.

For a more extensive comparison between solutions, we explored the key parameter of each solution and show the obtained results in Fig. 11. In the methods of *start-gap* and *security-refresh*, we varied their swapping periods (in terms of write counts). Thus, as the swapping period increases, lifetime extension and swapping overhead decrease. In our method, we varied the threshold of Bloom filter. Therefore, as the threshold value increases, lifetime extension and swapping overhead decrease.

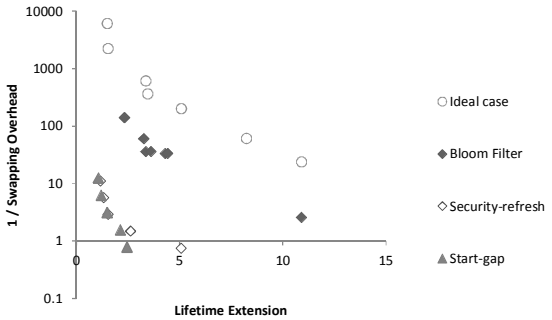

Fig. 11. Comparison with previous methods

5.3 Sensitivity Analysis

We explored the design space of our method by varying three parameters as shown in Fig. 12: Bloom filter size (the number of

Bloom filter entries), hot-cold list size and threshold value. In the first case, we fixed the hot-cold list size at 256, and applied policy 1 for dynamic threshold. Then, we varied the Bloom filter size from 128 to 512 entries (the results denoted with 'counter' in Fig. 12). With the larger Bloom filter, the lifetime extension ratio increases and swapping overhead decreases. It is because there are less false positives. In the second case, we fixed the Bloom filter counter size at 128, applied policy 1 and varied the hot-cold list size from 64 to 256 (the results denoted with 'list' in Fig. 12). The results in this case were slightly inferior to the previous cases of changing the Bloom filter size. It is because the false positive problem is significant in these cases (since the Bloom filter size is small). Thus, increasing the hot-cold list size is not so effective as increasing the Bloom filter size. Finally, we fixed the Bloom filter size at 256, fixed the hot-cold list size at 512 and varied the threshold value from 1000 to 8000. At low threshold values (e.g., 1000), both lifetime extension ratio and swapping overhead were at their largest levels. Lower threshold values result in frequent swappings thereby contributing to the flattening of writes. However, it increases swapping overhead as shown in Fig. 12.

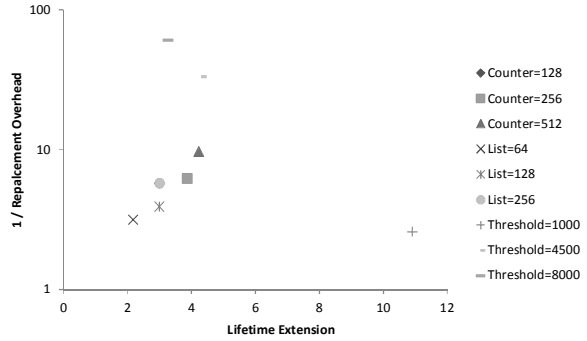

Fig. 12. Sensitivity analysis

5.4 Hot-Cold List Management Policy

In order to evaluate the effectiveness of three-tier hot-cold list, we compare three management policies: First-in-First-Out (FIFO), Least Recently Used (LRU) and the three-tier scheme (Section 4.3). We fixed the other parameters of our method at the best-performing values in general: the Bloom filter size is 256, the hot-cold list size is 256 and both polices 1 and 2 are used. Fig. 13 shows that the three-tier scheme increases lifetime extension ratio by 50.3%~87.7% and decreases swapping overhead by 73.7~79.3% over the FIFO or LRU policies.

6. Conclusion

Phase change memory (PCM) is a promising memory technology. However, one of the main drawbacks of PCM is the write endurance problem. To address this problem, we proposed a new wear-leveling method based on Bloom filters in order to reduce the space overhead of dynamic wear leveling. The proposed method gives 2.18~2.30 times improvement in lifetime while reducing swapping overhead by 58%~97% compared with existing methods while incurring a small area overhead of 0.3%.

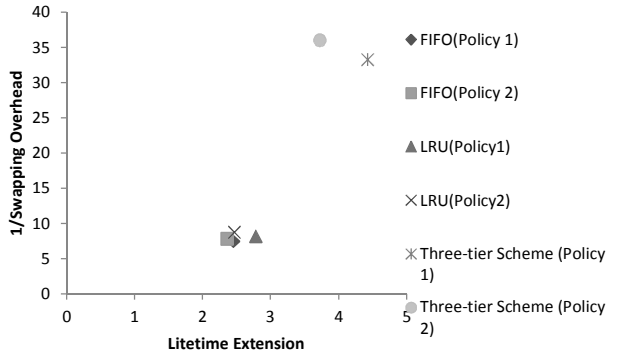

Fig. 13. Comparison of management policies

7. Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology, IC Design Education Center (IDEC) and Postech Information Research Laboratories (PIRL).

8. References

[1] R. Freitas and W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of R. and D.*, 52(4/5):439–447, 2008.

[2] M. Qureshi, V. Srinivasan and J. Rivers, "Scalable high performance main memory system using phase-change memory technology," *In ISCA-36*, 2009.

[3] B. Lee et al., "Architecting phase change memory as a scalable DRAM alternative," *In ISCA-36*, 2009.

[4] *International Technology Roadmap for Semiconductors*, ITRS 2007.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Tran. Commun. ACM*, 13(7):422–426, 1970.

[6] M. K. Qureshi et al., "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," *In MICRO-42*, 2009.

[7] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *In ISCA-37*, 2010.

[8] P. Zhou, B. Zhao, J. Yang and Y. Zhang, "A durable and energe efficient main memory," *In Proceeding of the International Symposium on Computer Architecture*, 2009.

[9] A. Ban and R. Hasharon, "Wear leveling of static areas in flash memory," U.S. Patent Number 6,732,221, 2004.

[10] A. Ben-Aroya and S. Toledo, "Competitive analysis of flash-memory algorithms," *In ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 100–111, 2006.

[11] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surv.,* 37(2)138–163, 2005.

[12] T. Kgil, D. Roberts and T. Mudge, "Improving nand flash based disk caches," *In ISCA '08: Proceedings of the 35th annual international symposium on Computer architecture*, pages 327–338, 2008.

[13] J. Dong, L. Zhang, Y. Han, Y.Wang, and X. Li, "Wear rate leveling : lifetime enhancement of PRAM with endurance variation," *In DAC 2011*, pp. 972-977, June 5-10, 2011.

[14] S. Li, et al., "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," Proc. MICRO, 2009.