ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.22, 1–11

Flexible block management with data migration wear-leveling algorithm for phase change memory

Mi Zhou[1,2a)], **Xiaogang Chen**[1b)], **Shunfen Li**[1], **Yueqing Wang**[1,2], **Yifeng Chen**[1], **Gezi Li**[1,2], **Yuchan Wang**[1,2], **and Zhitang Song**[1]

[1] *State Key Laboratory of Functional Materials for Informatics, Shanghai Institute of Micro-system and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China*

[2] *University of Chinese Academy of Sciences, Beijing 100080, China*

a) *mizhou@mail.sim.ac.cn*

b) *chenxg@mail.sim.ac.cn*

Abstract: Phase change memory (PCM) is regarded as a powerful competitor for future non-volatile memory applications. However, a key drawback is its limited write endurance. This paper proposes a flexible block management method with data migration wear-leveling algorithm (FBDM). The proposed method divides blocks into two halves meanwhile blocks can be split and merged flexibly. Data migration wear-leveling algorithm has been presented to extend the life of PCM. We simulated our method using different traces and compare it with previous methods. Simulation results show that the proposed method outperforms comparison methods in terms of wear evenness and overhead reduction.

Keywords: phase change memory, wear-leveling, endurance, block management

Classification: Storage technology

References

[1] R. Bianchini and R. Rajamony: Computer **37** [11] (2004) 68. DOI:10.1109/MC.2004.217

[2] R. F. Freitas and W. W. Wilcke: IBM J. Res. Develop. **52** (2008) 439. DOI:10.1147/rd.524.0439

[3] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan and R. S. Shenoy: IBM J. Res. Develop. **52** (2008) 449. DOI:10.1147/rd.524.0449

[4] S. R. Ovshinsky: Phys. Rev. Lett. **21** (1968) 1450. DOI:10.1103/PhysRevLett.21.1450

[5] T. Siegrist, P. Merkelbach and M. Wuttig: Annu. Rev. Condens. Matter Phys. **3** (2012) 215. DOI:10.1146/annurev-conmatphys-020911-125105

[6] B. Gleixner, F. Pellizzer and R. Bez: 2009 10th Annual Non-Volatile Memory Technology Symposium (NVMTS) (2009). DOI:10.1109/NVMT.2009.5429783

[7] E. Ipek, J. Condit, E. B. Nightingale, D. Burger and T. Moscibroda: ACM SIGARCH Comput. Archit. News **38** [1] (2010) 3. DOI:10.1145/1735970.1736023

[8] P. Zhou, B. Zhao, J. Yang and Y. Zhang: ACM SIGARCH Comput. Archit. News

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

37 [3] (2009) 14. DOI:10.1145/1555815.1555759
[9] M. K. Qureshi, J. Keridis, M. Franceschini, V. Srinivasan, L. Lastras and B. Abali: Proc. of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (2009) 14. DOI:10.1145/1669112.1669117
[10] C. H. Chen, P. C. Hsiu, T. W. Kuo, C. L. Yang and C. Y. Wang: 2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC) (2012) 453.
[11] J. S. Yun, S. G. Lee and S. J. Yoo: Proc. of the Conference on Design, Automation and Test in Europe (2012) 1513. DOI:10.1109/DATE.2012.6176713
[12] T. S. Chung, D. J. Park, S. Park, D. H. Lee, S. W. Lee and H. J. Song: *Embedded and Ubiquitous Computing* (Springer, Berlin, Heidelberg, 2006) 394.
[13] S. Lee, D. Shin, Y. J. Kim and J. Kim: Oper. Syst. Rev. **42** [6] (2008) 36. DOI:10.1145/1453775.1453783
[14] Z. W. Qin, Y. Wang, D. Liu and Z. L. Shao: Proc. of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (2010) 173.
[15] A. Gupta, Y. Kim and B. Urgaonkar: ASPLOS (2009) 229. DOI:10.1145/1508244.1508271
[16] M. K. Qureshi, V. Srinivasean and J. A. Rivers: ACM SIGARCH Comput. Archit. News **37** [3] (2009) 24. DOI:10.1145/1555815.1555760
[17] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem and D. Mosse: Proc. of the Conference on Design, Automation and Test in Europe (2010) 914. DOI:10.1109/DATE.2010.5456923
[18] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras and B. Abali: MICRO-42 Annual IEEE/ACM International Symposium (2009) 14.
[19] N. Agrawal, V. Prabhakaran, T. Wobbe, J. D. Davis, M. S. Manasse and R. Panigrahy: USENIX Annual Technical Conference (2008) 57.
[20] Websearch Trace from UMass Trace Repository: http://traces.cs.umass.edu/.

1 Introduction

In the past few decades, energy efficiency has become a critical design issue for storage systems as well as for servers [1]. Many low power memory systems have been proposed to reduce energy consumption, such as Storage Class Memory (SCM). SCM is a new class of memory replacing rotating mechanical storage with solid-state, nonvolatile RAM [2]. Using SCM as a disk drive replacement blurs the distinction between memory and storage, giving us new possibilities to save energy and simplify system configurations [3]. Various RAMs have been developed as potential candidates for SCM, such as phase change memory (PCM). PCM is a type of resistive memory that uses Ge, Sb, Te and other materials to produce phase change in atomic structures resulting in changes in resistivity [4]. Many studies have shown that PCM can give benefits due to its low energy, good read performance, non-volatility and bit addressability [5]. However, PCM suffers from a limited number of writes to each cell. PCM devices are expected to last for about 10^7–10^8 write cycles per cell [6, 7]. Although the endurance of a PCM cell is higher than a Flash memory cell, wear-leveling technology is still needed.

Wear-leveling is an effective way to distributes writes evenly throughout the whole memory space and improve the lifetime of storage systems. In recent years, a number of promising wear-leveling mechanisms have been proposed to improve the endurance of PCM. Segment swapping searches for the most frequently written

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

segment periodically and swaps them [8]. Start-gap method moves one line from its location to a neighboring location every η writes to main memory [9]. Bucket-based wear-leveling uses a number of buckets linked in a circular format to implement different distances so that a young page can be acquired immediately without searching pages. Array-based wear-leveling reduces the overheads involved in maintaining sophisticated data structures at the cost of a limited search overhead [10]. Bloom filter-based wear-leveling reduces the area overhead of write count information by utilizing the Bloom filter and the runtime overhead of sorting to find the hot address by utilizing an efficient hot-cold list management [11]. However, most of them are based on Flash. The storage space of Flash consists of physical blocks, and each block contains certain number of sectors. Before rewriting a block with new data, the entire block needs to be erased. When upper layers sending a request to write one sector belonging to a block, the block write cycle will increase even though other sectors in the same block haven't been written. Thus the entire block may be marked as worn out even though only few bytes may actually be worn out [12]. This leads to unnecessary statistic waste. Unlike Flash, memory cells in PCM can be modified one cell at a time and there is no need to erase a block before rewriting it. In order to manage PCM easily, most existing wear-leveling algorithms are block-level wear-leveling. They equally divide the memory space into blocks and record the write cycle of each block, there by failing to exploit the full lifetime potential of PCM, making block level wear-leveling a sub-optimal solution. Managing PCM on the unit of sector can address this problem. However, it is unsuitable for large sized PCM due to the large address mapping table [13].

In this paper, we propose a low statistic waste solution by applying the flexible block management. The basic idea is to split and merge blocks when satisfying a certain condition. Split operations help to separate regions which have large write cycles from regions with small write cycles in one block. To avoid producing too many memory fragments, adjacent blocks can be merged into one unity. A novel PCM Translation Layer (PTL) dynamically translates logical address to physical address or vice versa with line as units. Based on the flexible block management, data migration wear-leveling algorithm is presented to protect blocks with high write frequency. We compare the proposed approaches with swap-based WL, start-gap WL and also estimate two baseline schemes, no wear-leveling and an ideal case. The results of experiments demonstrate that the proposed approaches are very effective in improving the endurance of PCM.

2 Flexible block management

Conventional block management divides storage space into blocks with equal size, and each block consists of a fixed number of sectors [14]. A block is the smallest unit involved in erase operations while read and write operations are done in sectors. Considering the random read and write property of PCM, dividing PCM into blocks may decrease the system flexibility and cause statistic waste of write cycles. However, if we manage PCM storage space in sectors, the system overhead will increase by a large amount. To strike a balance between performance and system overhead, we adopt flexible block management.

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

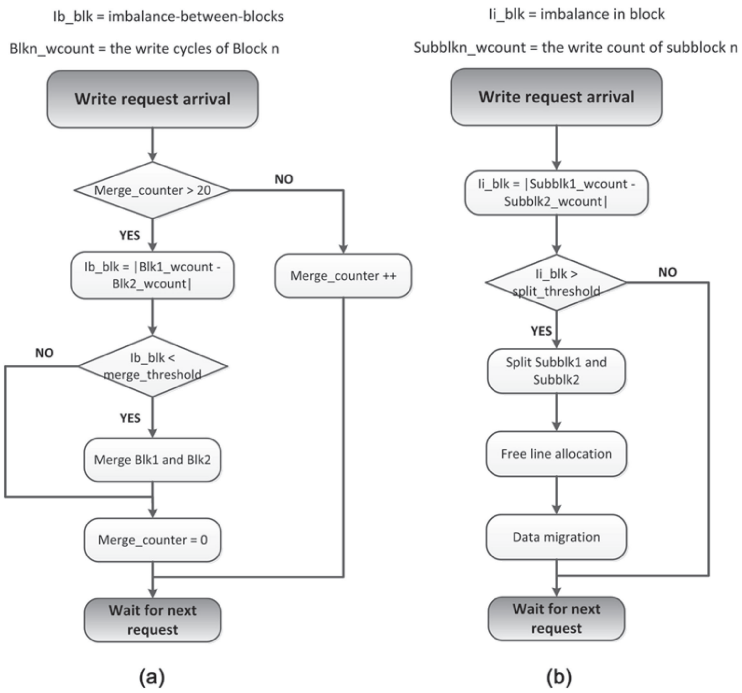

Fig. 1. Flow chart of merge operation (a) and split operation (b).

Flexible block management initially divides PCM using the same method as conventional block management. The difference is that in our proposal, each block is divided into two halves called sub-block and each sub-block has a counter to record its write cycles. Sub-blocks are used to evaluate the imbalance within a block. Block write cycles is defined as the larger one of the two sub-block write cycles and split/merge thresholds are set to determine when the split/merge operations should be triggered. We measure the imbalance-in-block by the absolute difference of two sub-blocks' write cycles. Great imbalance-in-block means the uneven distribution of write operations within block. On the arrival of a write request with a logical address, its corresponding physical block will be checked. If the imbalance-in-block of this block is greater than the split threshold, the block will be split and each of its sub-blocks will become an independent block. Similarly, the imbalance-between-block can be calculated by the absolute difference of two neighboring blocks' write cycle. Small imbalance-between-block indicates that the two neighboring blocks have approximate write cycles and they can be merged as a whole block to prevent excess memory fragments. Flexible block management checks the imbalance-between-block periodically and compares it with the merge threshold. If the imbalance-between-block is smaller than merge threshold, the two blocks will be merged together as a whole block. The flow charts of merge and split operations are shown as Fig. 1.

3 Line-level mapping

Mapping resolutions have direct impact on system performance [15]. Block-level mapping requires only small mapping structures. However, it is not applicable for flexible block management since the sizes of blocks change along with split/merge operations. Sector-level mapping better handles random requests and applies to our

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

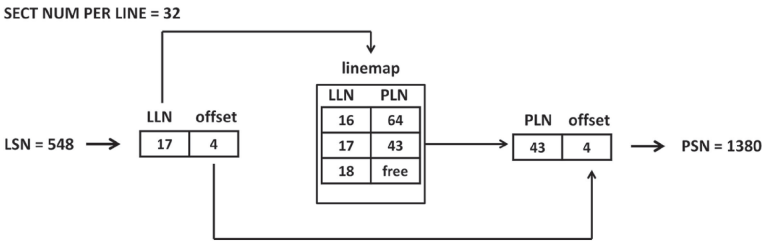

Fig. 2. The line-level mapping scheme.

method, but requires large mapping structures. In this paper, we use line-level mapping as the address translation scheme of PTL in PCM. Line-level mapping groups 32 consecutive logical sectors as logical lines as large as physical lines and a block contains 8 lines at the beginning. It maps logical lines to physical lines on a one-to-one basis using a line-level mapping table. In line-level mapping, a logical sector number (LSN) is divided by the number of sectors in a line to obtain its logical line number (LLN) and line offset, where the LLN is the quotient and the line offset is the remainder of the division. A line-level mapping table redirects the write operations on logical line (LLN) to a physical line (PLN). When a logical address is mapped to a new line, the offset is fixed and only the line number is changed. On the arrival of a write request with a logical address, the line-level mapping table will be checked. If the LLN does not exist, an entry will be created and the mapping table is set up accordingly. Fig. 2 shows how to translate LSN to PSN in the line-level mapping scheme. When a request for a logical sector 548 comes to the PCM, it first calculates its logical line 17 and line offset 4. Then, the logical line 17 accesses the mapping table to translate it to the physical line 43. Note that the size of a line is adjustable and mapping resolution can be easily controlled to reach a tradeoff between mapping structure and space overhead.

4 Wear-leveling

Flexible block management provides a good fundamental architecture for PCM wear-leveling. On the basis of flexible block management, our wear-leveling algorithms include three portions: free-line-allocation, data migration and protect queue.

4.1 Free-line-allocation

Algorithm 1: Free Line Allocation

Input: A free line allocation request arrives, and the free_line_allocation is invoked.

Output: *PLN*, allocated free line number

```
1  while free_queue[free_queue_index] ≠ NULL
2    if free_queue[free_queue_index].is_free=1 then
3      PLN←free_queue[free_queue_index].pln
4      if free_queue_index<k then
5        free_queue_index++
6      else
```


DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

```
7            free_queue_index ← 0
8          end if
9          break
10       else
11         free_queue[free_queue_index] point to the next line
12         if free_queue[free_queue_index].pln = max line number
           then
13           free_queue[free_queue_index].pln ← 0
14         end if
15       end if
16     end while
17     return PLN
```

Free-line-allocation always returns a free line when needed during the execution of a process or when invoked by the OS. Procedure free-line-allocation, as shown in Algorithm 1, employs a pointer array size of k, an external pointer pointing to the elements in the array and a free-line queue. After initialization, the storage space is divided into k equal sections. The k pointers in the pointer array points to the first line of the corresponding sections. When a free line is needed, the OS returns a page according to the location of pointers. Once a free line has been acquired, the pointer will be updated and added to the end of the free-line queue. Current pointer will move to the next line and the external pointer will point to the next element in the pointer array. Note that this procedure distributes write operations evenly throughout the whole storage space.

4.2 Data migration

Data migration focuses on the wear of elder sub-blocks because they retire earlier than junior sub-blocks. It relocates data in a badly worn sub-block to a less worn block when splitting and then isolates the worn sub-block by adding it to a protect queue. If a block with high imbalance-in-block satisfies the split criterion, a split operation is evoked to separate elder sub-blocks from junior sub-blocks. All valid data in the worse worn sub-block is copied into free lines acquired from the free-line-allocation mechanism.

4.3 Protect queue

Protect queue is a First-In-First-Out queue. The design objective of protect queue is to protect blocks with high write cycles from further aging. In addition, protect queue also takes efforts to prevent unlimited split operations. The details of protect queue are described in Fig. 3. On condition that a block reaches the minimum size we stipulated, valid data in the block needs to be migrated to a less worn block. When data migration accomplishes, current block becomes a free block and joins the end of the protect queue. Blocks in the protect queue will stay unusable until the queue is full and starts to release blocks. The length of protect queue determines how long a block will be protected.

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

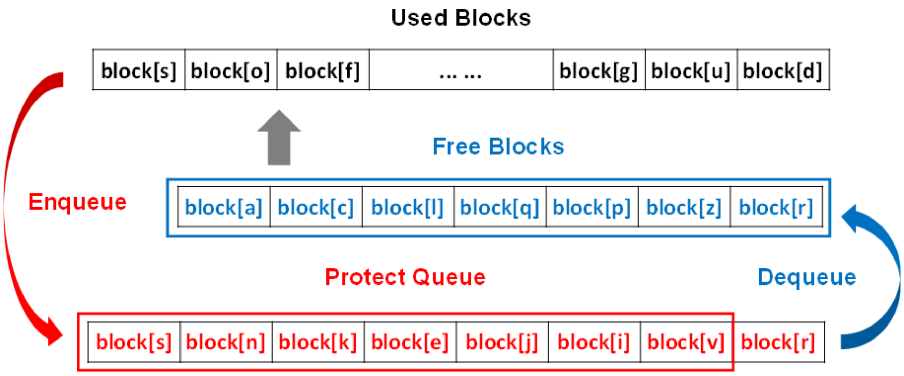

Fig. 3. The protect queue scheme.

5 Experimental setup and performance metrics

We evaluate the performance of the proposed FBDM in terms of lifetime and overhead. The lifetime metric is based on the maximum value and the standard deviation of write cycles, and the overhead is based on the memory space occupancy due to wear-leveling. For storage system with write traffic of B MBps, a PCM of size S MB and a cell endurance of W_{max}, the duration of an ideal case is given by the following equation [16]:

$$SystemLifetime = \frac{W_{\max} \bullet S}{B} Second \quad (1)$$

Lifetime and overhead have to be assessed together in order to obtain a qualitative judgment about the efficacy of the algorithms. We compare two latest wear-leveling schemes: swap-based WL [17] and start-gap WL [18]. Swap-based WL swaps the page currently being written with a randomly selected page for every 512 writes to PCM pages. Start-gap WL performs wear-leveling by periodically moving each line to its neighboring location, regardless of the write traffic to the line. To better identify the motivation for the adoption of wear-leveling, we estimate two baseline schemes: no wear-leveling and an ideal case when all the writes are spread evenly over the PCM storage space. Furthermore, swap-based WL based on flexible block management (FB swap-based WL) is also realized to compare wear-leveling algorithms.

We extract key parameters from the real PCM chips and build a simulation system to run the proposed algorithm. Merge_counter, split_threshold, merge_threshold and the length of the protect queue have a great influence on the performance of FBDM. We experiment with different values of them and select the optimal results. Our simulation environment is developed on the basis of the DiskSim simulator from the CMU Parallel Data Lab [19]. Disksim emulates a hierarchy of storage components such as buses and controllers as well as disks. It does not specifically support simulation of solid state disks, but its extensibility made it a good vehicle for customization. We modified DiskSim to support multiple interfaces so as to connect different storage devices. The system architecture is shown in Fig. 4. The wear-leveling and mapping schemes are packaged as PCM translation layer (PTL) and loaded in the Storage Device (SD) layer.

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

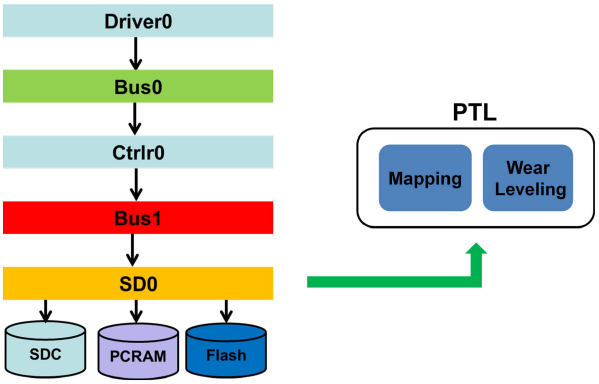

Fig. 4. PCM simulator system architecture.

To collect the reliable parameter for our simulation system, a test on a 32 kbit PCM chip has been performed. The 200–500 ns SET and 200 ns RESET resistance distributions are shown in Fig. 5(a). It can be inferred from the trend of resistance that the resistance window between the SET and RESET state is wide enough to distinguish between the two states when SET pulse is wider than 500 ns. Fig. 5(b) shows the read hit rate increases as the read period increases and it reaches 100% when read period exceeds 165 ns. According to the test results, a series of operation parameters are incorporated in our simulation system, as shown in Table I. One 512 MB PCM memory is employed with 512 Byte per sector and 10^6 write cycles.

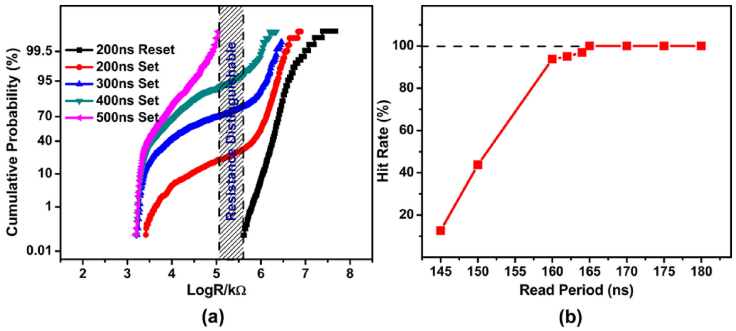

Fig. 5. (a) The resistance distribution of PCM cells. (b) The read hit rate of PCM cells.

Table I. Experiment setup

Memory type	Memory Unit Size		Waveform Setting				Operation Pulse			
	Memory Size (MB)	Sect (Byte)	Write		Read		SET		RESET	
			Latency (ns)	Interval (ns)	Latency (ns)	Interval (ns)	Width (ns)	Voltage (V)	Width (ns)	Voltage (V)
PCM	512	512	800	500	200	200	500	3.3	200	3.3

6 Workloads

We use a mixture of real-world and synthetic traces to study the impact of FBDM on a wide spectrum of workloads. We experiment using a multimedia trace extracted from DiskMon developed by Microsoft, running several applications

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

such as documents editors, music players and games. We also employ a write-dominant I/O trace from simulation software running at a laptop. Also, we use a modified Web Search engine trace made available by SPC [20]. Finally, we generated an attack trace by the TraceGen, a program developed to generate traces that meet our requirements. The attack trace writes only five thousand sectors in two million requests, providing an extreme case for testing. The four traces can basically cover various daily applications and perform a reliable evaluate on the FBDM algorithm.

7 Experimental results

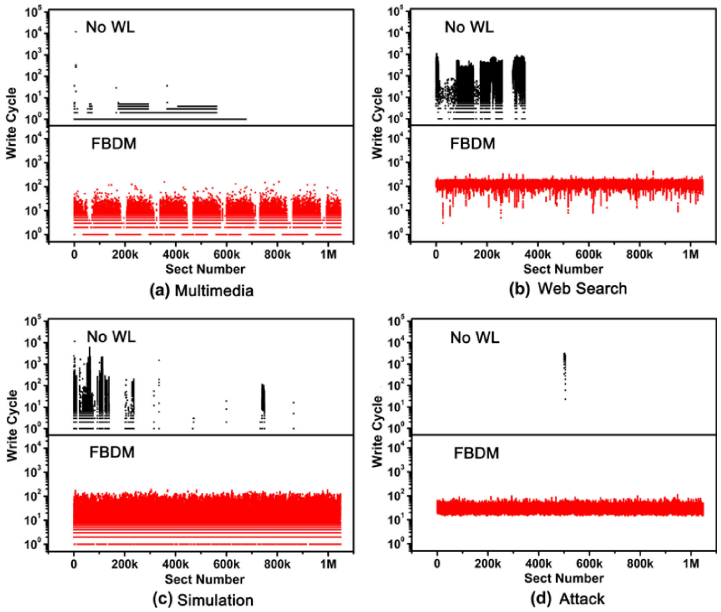

Fig. 6. The distribution of write cycles of No WL system and FBDM system for (a) Multimedia, (b) Web Search, (c) Simulation and (d) Attack traces.

Fig. 6 are write cycle distributions of four traces and the corresponding distributions after FBDM. The upper parts shows that write request of the four traces are uneven distributed and some sectors have been written frequently. The bottom parts shows that write cycles are more evenly distributed throughout storage space by applying FBDM. Fig. 7 shows the cumulative probability distribution of different traces under four algorithms. We see that in flexible block system write cycle distributions concentrate in a relatively narrow range, which indicates a higher evenness. The maximum write cycle of FBDM decreases 1–2 orders of magnitude compared with the NO WL system. Standard deviation of write cycles under different traces and different wear-leveling algorithms is shown as Fig. 8. The lower value of standard deviation means the better performance of wear-leveling. This implies that FBDM achieves better wear evenness than FB swap-based WL and start-gap WL and flexible block management's standard deviations are much smaller than that of traditional block management.

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

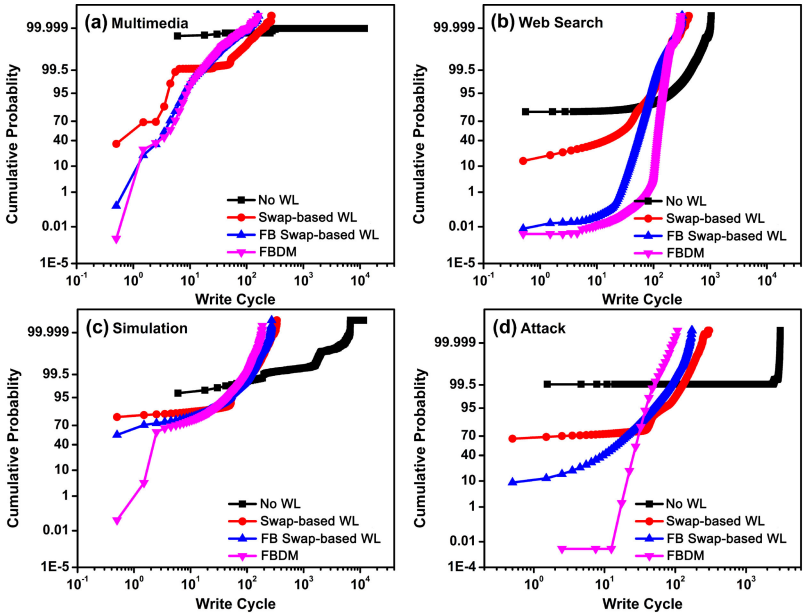

Fig. 7. Cumulative probabilities for (a) Multimedia, (b) Web Search, (c) Simulation and (d) Attack trace.

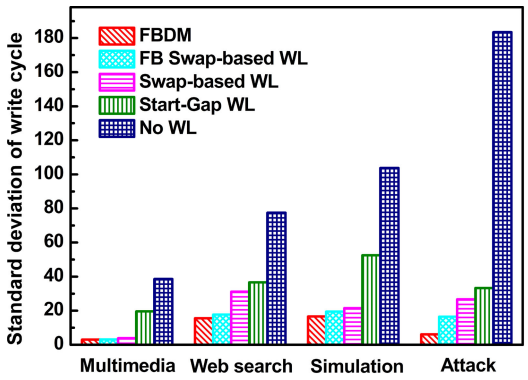

Fig. 8. Standard deviation of write cycle.

The improvement is remarkable under the attack trace, meaning that extreme conditions can be well handled by FBDM. We estimate the lifetime in years based on the maximum write cycle. Fig. 9 shows the lifetime achieved by the invested approaches under various traces. We observe that the FBDM extends the memory lifetime by 3–75 times over no wear-leveling system, 1.6 times over swap-based WL and at least 2 times over start-gap WL. Fig. 10 shows the overhead incurred by the invested approaches. We can calculate the required space overhead by line number, block number and mapping table. NO WL system is set as the normalized reference. Since the start-gap WL requires storage only for two registers, the overhead can be neglected here. The results show that the overheads of flexible block management systems are obviously lower than that incurred by swap-based WL. Reason for the low overhead is that in FB systems neighboring blocks can be merged as one block so as to maintain the block number a certain amount.

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014

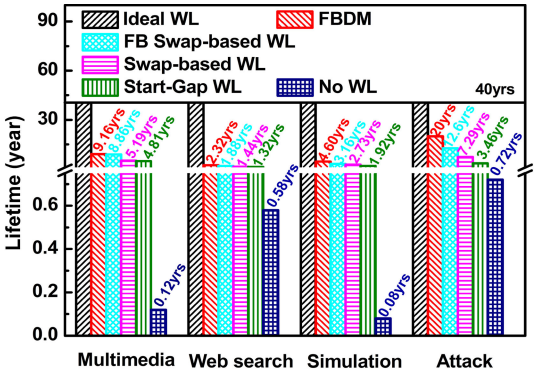

Fig. 9. The lifetime under different traces.

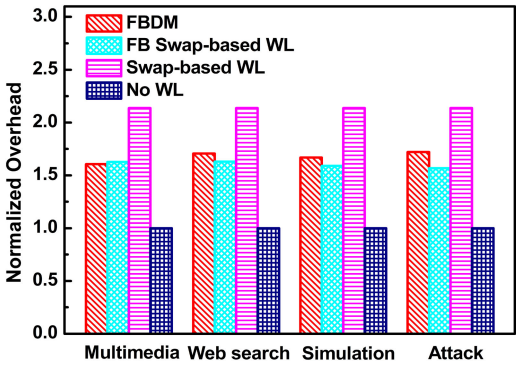

Fig. 10. The normalized overhead under different traces.

8 Conclusion

Phase Change Memory is a promising non-volatile technology. However, one of the main challenges in a PCM system is the write endurance problem. To address this problem, we proposed a new wear-leveling method based on flexible block management in order to reduce the statistic waste. We conduct extensive experiments under popular benchmarks to evaluate the endurance improvement as well as the extra overhead incurred. The experimental results show that the approaches could achieve 1.6–75 times improvement while incurring a negligible area overhead. In the future, we plan to explore experimental configurations to realize optimal wear-leveling.

Acknowledgments

Supported by National Key Basic Research Program of China (2011CB932804, 2013CBA01900, 2011CBA00607, 2010CB934300), the "Strategic Priority Research Program" of the Chinese Academy of Sciences (XDA09020402), National Integrate Circuit Research Program of China (2009ZX02023-003), National Natural Science Foundation of China (61076121, 61176122, 61106001, 61261160500, 61376006), Science and Technology Council of Shanghai (11DZ2261000, 12nm0503701, 12QA1403900, 13ZR1447200, 13DZ2295700).

DOI: 10.1587/elex.11.20140924
Received September 26, 2014
Accepted October 17, 2014
Publicized November 7, 2014
Copyedited November 25, 2014