A Block-Level Flash Memory Management Scheme for Reducing Write Activities in PCM-based Embedded Systems

Duo Liu, Tianzheng Wang, Yi Wang, Zhiwei Qin and Zili Shao
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
cszlshao@comp.polyu.edu.hk

***Abstract*—This paper targets at an embedded system with phase change memory (PCM) and NAND flash memory. Although PCM is a promising main memory alternative and is recently introduced to embedded system designs, its endurance keeps drifting down and greatly limits the lifetime of the whole system. Therefore, this paper presents a block-level flash memory management scheme, WAB-FTL, to effectively manage NAND flash memory while reducing write activities of the PCM-based embedded systems. The basic idea is to preserve each bit in flash mapping table hosted by PCM from being inverted frequently during the process of mapping table update. To achieve this, a new merge strategy is adopted in WAB-FTL to delay the mapping table update, and a tiny mapping buffer is used for caching frequently updated mapping records. Experimental results based on Android traces show that WAB-FTL can effectively reduce write activities when compared with the baseline scheme.**

***Index Terms*—Phase change memory, NAND flash memory, flash translation layer, endurance, write activity.**

I. INTRODUCTION

Due to its high density, in-place update, and low standby power, phase change memory (PCM) is considered as a DRAM alternative and has been used as a main memory with a small-sized DRAM cache in embedded systems [1–3]. However, compared to DRAM, PCM can only sustain limited write operations (10^6 to 10^8 bit flips per cell) [4], so it is necessary to reduce write activities in PCM to enhance its lifetime. On the other hand, NAND flash memory is widely used as a secondary storage and has been integrated into PCM-based embedded systems [5, 6]. How to avoid a fast worn-out of such emerging embedded systems and effectively manage NAND flash memory should be taken into account. Extensive work recently has been done to reduce write activities for enhancing the lifetime of PCM [2, 3, 6–10]. However, none of them considers write activities caused by the management procedure of NAND flash memory. Therefore, this paper focuses on exploring a write-activity-aware NAND flash memory management scheme in PCM-based embedded systems to enhance the lifetime of the entire system.

To manage NAND flash memory, flash translation layer (FTL) is designed to emulate it as a disk drive, by mapping logical addresses to physical addresses at a granularity of page-level or block-level [11–15]. Following I/O requests, an FTL mapping table is employed to keep track of the continually updated mapping records. To provide fast lookup, FTL mapping table is usually loaded into main memory after system is booted, and put back to NAND flash memory once the system is shut down. In DRAM-based main memory, the most-updated FTL mapping table can be lost due to power failure. However, as PCM is non-volatile, FTL mapping table can be kept into PCM-based main memory permanently without considering power failure. Therefore, Kim et al. [5] propose a page-level FTL, *h*FTL, whose page-level FTL mapping table is kept in PCM and user data is stored in NAND flash memory. Nevertheless, *h*FTL does not consider write activities imposed in PCM because of the frequently updated FTL mapping table, which may lead to a shortened PCM lifetime.

Though a page-level-based FTL technique is proposed for improving the lifetime of PCM-based embedded systems [16], page-level FTL requires significant memory requirement, so it may not be applicable for current PCM chips whose capacity is reported as 128Mb [17], e.g., the page-level mapping table of a 1GB flash memory occupies approximately 12.5% space of the 128Mb Micron P5Q PCM. Thus block-level FTL with much less memory requirement is more applicable for PCM-based embedded systems[13, 14]. Several block-level FTL schemes have been proposed, but none of them considers write activities imposed to PCM. Since the lifetime of PCM is mainly determined by the maximum number of bit flips in each PCM cell, no matter how smaller the block-level mapping table is, it is important to reduce the maximum number of bit flips in each PCM cell to enhance the reliability of the entire system. These observations motivate us to propose a block-level FTL scheme to reduce write activities in PCM, such that the lifetime of the entire PCM-based embedded systems is enhanced.

In this paper, we propose a **W**rite-**A**ctivity-aware **B**lock-level **FTL** scheme, called **WAB-FTL**, to reduce write activities in PCM during the management procedure of NAND flash memory and, at the same time, to enhance the lifetime of the PCM-based embedded systems. Our basic idea is to preserve each bit in FTL mapping table, i.e., each bit in PCM cell, from being inverted frequently, during the update process of FTL mapping table, such that the maximum number of bit flips in each PCM cell is reduced and the lifetime of PCM is enhanced. To achieve this, a new merge strategy, called Lazy-Merge, is adopted in the proposed WAB-FTL to delay the mapping table update, and a tiny mapping buffer, called Cooling-Pool, is used for caching frequently updated mapping records. Then by utilizing a fine-grained hardware feature [7], in which a write can be eliminated if its designated PCM cell holds the same value. In WAB-FTL, we can actively choose a destination mapping slot, wherein the old mapping record has the minimum Hamming distance to the new mapping record, and then only update (flip) the bits distinct from that in the new mapping record. To the best of our knowledge, WAB-FTL is the first block-level flash memory management scheme proposed for reducing write activities in PCM-based embedded systems.

We conduct a series of experiments on a set of realistic I/O traces collected from Google Android™2.3. Compared with a well-known block-level FTL scheme, experimental results show that our technique achieves an average reduction of 80.76% and a maximum reduction of 82.61% in the maximum number of bit flips.

This paper makes the following contributions:

- We present for the first time a block-level flash memory management scheme to reduce write activities in PCM-based embedded systems for enhancing the PCM lifetime.
- We demonstrate the effectiveness of our technique by comparing with representative block-level FTL schemes using a set of realistic I/O workloads collected from Google Android™2.3.

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III presents our proposed WAB-FTL technique. Section IV reports the experimental results. Finally, in Section V, we conclude the paper.

II. BACKGROUND AND MOTIVATION

A. PCM-Based Embedded Systems

Fig. 1 shows a typical PCM-based embedded system, which consists of a hybrid PCM-based main memory and a NAND flash

978-3-9810801-8-6/DATE12/©2012 EDAA

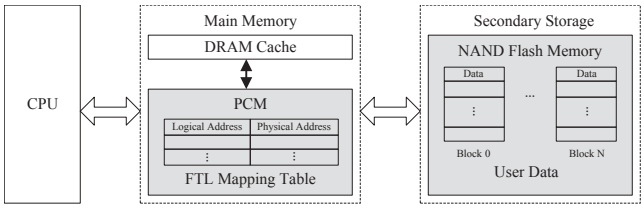

Fig. 1: PCM-based embedded system with NAND flash memory.

memory. To obtain a best capacity and latency, the hybrid main memory adopts a large-sized PCM and a small-sized DRAM. PCM acts as a main memory for maintaining frequently accessed OS pages and the FTL mapping table, while the DRAM acts as a cache and bridges the gap between PCM and the processor. In the system, NAND flash memory is employed as a secondary storage for storing user data that accessed by file systems. Following I/O requests, the mapping from logical address to physical address will be updated continually in FTL mapping table. So the FTL mapping table is the most heavily updated component in PCM and may shorten PCM lifetime if some unnecessary write activities are performed. Therefore, to avoid the lifetime degradation of PCM, it is necessary to make block-level FTL scheme write activity aware in PCM-based embedded systems.

B. The Baseline Scheme

In this section, we briefly revisit a well-known block-level FTL scheme, BL-FTL, which is widely used in embedded systems [14]. In BL-FTL, a logical page number (LPN) is divided by the number of pages in a block to obtain its logical block number (LBN) and block offset, where the LBN is the quotient, and the block offset is the remainder of the division. A block-level mapping table redirects the write operations on logical block (LBN) to a physical primary block (PPBN). For each primary block, only one physical replacement block (PRBN) is allocated to handle subsequent update operations. A write operation to an LPN is mapped to a page in a primary block first based on block offset, and subsequent update operations to the same LPN are written into the corresponding replacement block consecutively. Therefore, the most-updated content can be found by reading the replacement block backwards. If a replacement block is full, a merge operation (denoted by Full-Merge hereinafter) is evoked to reclaim the replacement block and its associated primary block, and all valid pages in the two blocks are copied into a new primary block.

An example of BL-FTL is shown in Fig. 2. To simplify the example, we assume each block has eight pages, and there are only two free blocks in the free block list. The address of pages/blocks is represented by binary number to demonstrate bit flips in mapping table. The I/O requests of write operations (w) are listed in Fig. 2(a). According to the I/O requests, Fig. 2(b) shows the status variation of the blocks in NAND flash memory, and Fig. 2(c) shows the bit flips occurred due to the update of block-level mapping table in PCM. As shown, for the first 12 requests, a primary block (PPBN #010) and its replacement block (PRBN #001) are allocated, so the corresponding mapping (010, 001) is recorded into the block-level mapping table. Once the replacement block (PRBN #001) is full, both of these two blocks (PPBN #010 and PRBN #001) are erased together and the valid pages are copied into a newly allocated primary block (PPBN #101). Meanwhile, the eased primary and replacement blocks are put into free block list for further use. For the remaining requests (13-20), they are served in a similar way. Finally, as shown in Fig. 2(c), the total number of bit flips caused by the update of mapping table is 12, and the maximum number of bit flips in each PCM cell is 2.

C. Motivation

In the motivational example, it is noticed that each bit used to represent the mapping record is inverted in a round trip (0→1→0)

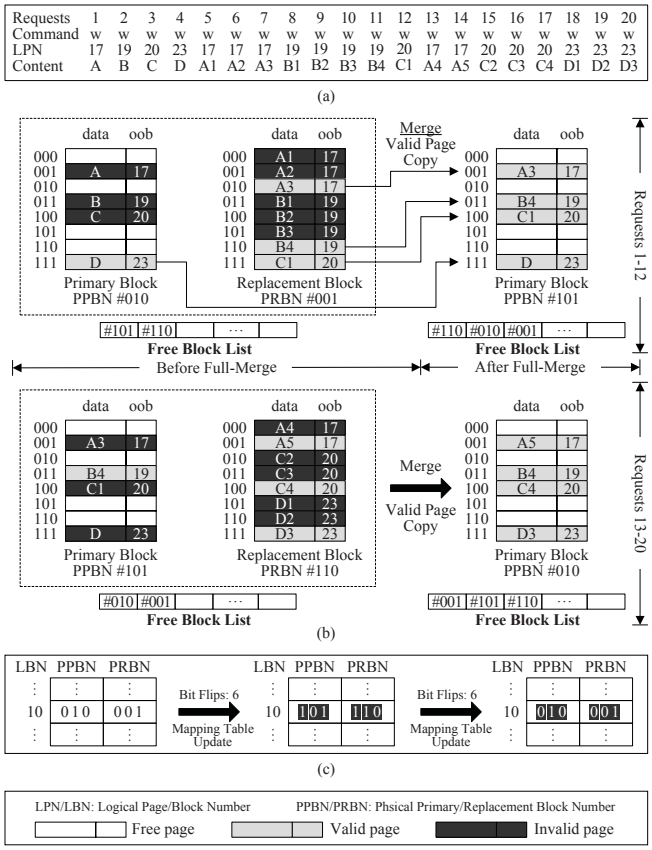

Fig. 2: Motivational example. (a) I/O access requests. (b) The status variation of blocks in NAND flash memory. (c) The bit flips caused by the update of block-level mapping table in PCM.

due to the update of mapping table. If this bit-flip pattern continually happens in realistic applications, then the lifetime of PCM will decrease faster. In addition, as the lifetime of PCM is mainly determined by the maximum number of bit flips in each cell, it is important to avoid unnecessary bit flips during the update of FTL mapping table. Once the maximum number of bit flips in each PCM cell is reduced, then the lifetime of PCM is enhanced. These observations motivate us to propose a write-activity-aware block-level FTL to reduce the maximum number of bit flips in PCM, such that the lifetime of the entire PCM-based embedded systems is improved.

III. WAB-FTL: Write-Activity-Aware Block-Level FTL

A. Overview

The basic idea of WAB-FTL is to preserve each bit in FTL mapping table hosted by PCM from being inverted frequently. Thus, to make WAB-FTL write activity aware, we develop the following techniques:

- A new merge strategy, named Lazy-Merge, is proposed to delay the update of mapping records in FTL mapping table. With Lazy-Merge, a replacement block will be erased if it is full, but its associated primary block with corresponding mapping record is preserved until no free blocks is left.
- A tiny buffer, named Cooling-Pool, is proposed to reduce write activities in block-level mapping table, by caching the frequently updated mapping records which may impose significant number of bit flips in PCM.

B. WAB-FTL with Lazy-Merge Strategy

In WAB-FTL, Lazy-Merge strategy is a simple yet effective technique to reduce write activities occurred during the update process

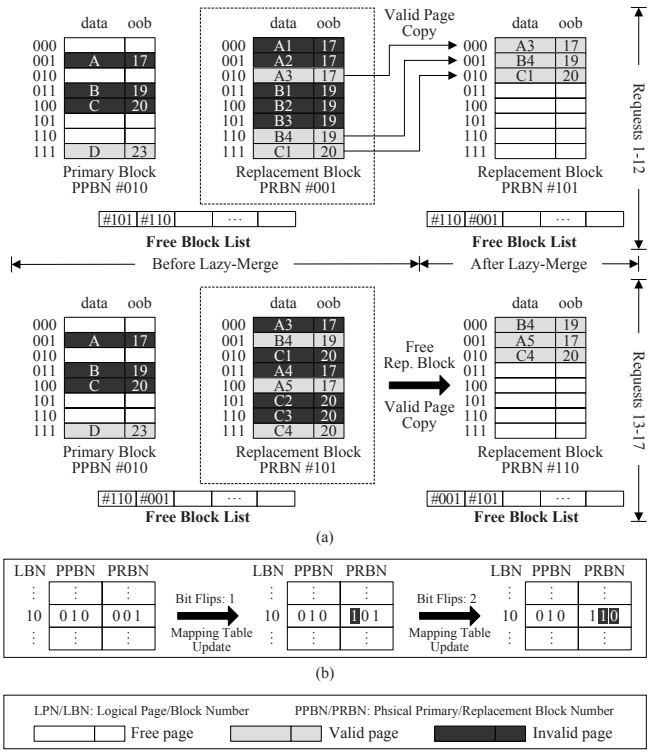

Fig. 3: Example of WAB-FTL. (a) The status variation of blocks in NAND flash memory. (b) The bit flips caused by the update of block-level mapping table in PCM.

of block-level mapping table. Unlike Full-Merge, we propose Lazy-Merge strategy, by which we only erase the replacement block and preserve its associated primary block from being erased. It is noticed that an update to the mapping record of the primary block is avoided, such that some write activities to PCM are reduced. Moreover, the block erase counts can also be reduced if the primary block is preserved in a merge operation.

In our Lazy-Merge strategy, when a replacement block is erased, all valid pages in the old replacement block will be copied into a new allocated replacement block, and the corresponding mapping record of the old replacement block will be updated by the new one. For the associated primary block, its corresponding mapping record in mapping table remains unchanged until the primary block is erased in a garbage collection for reclaiming more free blocks. Therefore, with Lazy-Merge strategy, lots of updates to the mapping records of the primary blocks are eliminated, and thus bit flips in each PCM cell are effectively reduced.

An example is illustrated to show Lazy-Merge in Fig. 3. Based on the same I/O requests and assumptions in Fig. 2, for the first 12 requests, the status variation of blocks and mapping table in Fig. 3 is exactly the same as that in motivational example. However, by adopting our Lazy-Merge strategy, we only erase replacement block (PRBN #001) and preserve the primary block (PPBN #010), and at the same time, copy the valid pages from the old replacement block (PRBN #001) to a new allocated replacement block (PRBN #101) in a consecutive order. Correspondingly, the PRBN in mapping table is updated with only one bit flip occurred. Then the new replacement block (PRBN #101) can be used to serve the rest requests (13-17). With our Lazy-Merge strategy, the remaining requests are served in a similar way. Finally, as shown in Fig. 3(b), the total number of bit flips caused by the update of mapping table is only 3, and the maximum number of bit flips is 1. This example shows

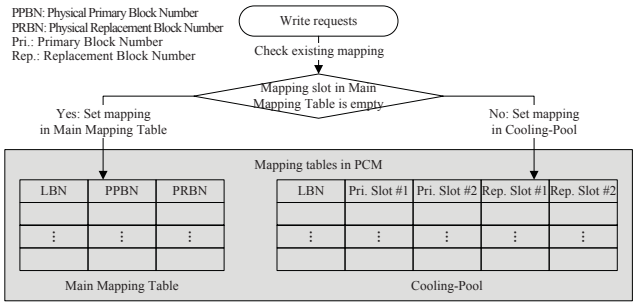

Fig. 4: WAB-FTL Management.

that our technique achieves a reduction of 75.00% (50.00%) in the total (maximum) number of bit flips compared to the motivational example. The example may not reflect realistic applications, however, the experimental results with realistic I/O traces in Section IV show that our approach can significantly reduce write activities in PCM.

C. WAB-FTL with Cooling-Pool

In WAB-FTL, by adopting Lazy-Merge, the mapping records of replacement blocks are updated more frequently than that of primary blocks. This motivates the design of Cooling-Pool for caching the frequently updated mapping records, to prohibit the PCM area with frequently updated mapping records from being wear out earlier. Fig. 4 shows the structure of WAB-FTL. As shown, in addition to the block-level mapping table (main mapping table), WAB-FTL employs a Cooling-Pool, in which multiple candidate mapping slots for primary blocks (Pri. Slots) and replacement blocks (Rep. Slot) are allocated, for caching the mapping records of frequently updated requests. In WAB-FTL, all new mapping records are first written into the main mapping table, and the following updated mapping records are written into the Cooling-Pool.

Algorithm III.1 shows the detailed process of WAB-FTL management. When the I/O requests arrive, WAB-FTL first checks if the corresponding LBN is mapped to a $PPBN$ in the main mapping table. If no mapping record is found, it means that this is a new write. WAB-FTL will allocate a new primary block and set the (LBN,

Algorithm III.1 The algorithm of WAB-FTL

Input: I/O requests.
Output: Map LBN to PBN.

```
1: Check current mapping state.
2: if There exists no (LBN, PPBN) mapping in main mapping table or Cooling-Pool then
3:     This is a new write, allocate a new primary block PPBN, and write the contents based on block offset.
4:     Add (LBN, PPBN) mapping into main mapping table.
5: end if
6: if There exists (LBN, PPBN) mapping in main mapping table or Cooling-Pool then
7:     This is an update.
8:     if There exists (LBN, PRBN) mapping in main mapping table or Cooling-Pool then
9:         if The number of free pages in replacement block ≤ number of update pages to be written then
10:            Allocate one new replacement block.
11:            if The (LBN, PRBN) mapping resides in main mapping table then
12:                Allocate one entry in Cooling-Pool.
13:            else
14:                Get (LBN, PRBN) entry in Cooling-Pool.
15:            end if
16:            Update (LBN, PRBN) mapping in the Cooling-Pool entry.
17:        end if
18:        Write the new contents to replacement block.
19:        Invalidate original pages that are updated in primary block (if any) or replacement block (if any).
20:    else
21:        Allocate one new replacement block and write the new contents.
22:        Add (LBN, PRBN) mapping into the replacement slot of Cooling-Pool.
23:    end if
24: end if
```

$PPBN$) mapping in main mapping table. Otherwise, it will further check whether there is $PRBN$ mapped for the incoming LBN. If so, and the replacement block is not full, updates will be written to the replacement block consecutively. If the replacement block is full, a new replacement block will be allocated, and a new (LBN, $PRBN$) mapping record will be added into the Cooling-Pool. Therefore, in Cooling-Pool, a mapping slot who incurs the minimum number of bit flips will be selected for accommodating the new $PRBN$ value. In case that no replacement block is allocated for the LBN, WAB-FTL will allocate a new replacement block, and the $PRBN$ will be written to the replacement block slot in main mapping table. When the Cooling-Pool is full, an entry that is not frequently updated will be selected as a victim for replacement, and the corresponding mapping records will be moved to the main mapping table. Note that the Cooling-Pool is extremely small, and its size is merely 1% of the main mapping table size. Therefore, the capacity overhead introduced by Cooling-Pool is acceptable when compared to its contribution of the write activities reduction in PCM.

IV. EVALUATION

To evaluate the effectiveness of WAB-FTL, we conduct a series of experiments and present the experimental results with analysis in this section. We compare and evaluate our proposed WAB-FTL scheme over a well-known block-level FTL scheme (BL-FTL), in terms of the maximum and total number of bit flips in PCM cells.

In this paper, we assume that the FTL mapping tables are stored in a single-level cell (SLC) PCM, and the user data is stored in a multi-level cell (MLC) NAND flash memory, which is widely used in embedded systems.

A. Experimental Setup

The evaluation is conducted by a trace-driven simulation. We have developed a simulator to evaluate BL-FTL and our WAB-FTL. Traces along with various NAND flash parameters, such as block size and page size, are fed into our simulator. The page size, number of pages in a block, and size of the OOB area for each page are set as 2 KB, 64, and 64 Bytes, respectively. Table I summarizes the configurations of Android Emulator and our simulation environment. The simulated system also features 128Mb PCM for storing mapping tables.

Four traces of I/O requests were collected from Google Android™2.3 with Android Emulator (included in Android SDK), as shown in Table II.

TABLE I: Experimental Setup

Android Emulator Configuration	CPU	ARM926EJ-S
	OS Kernel	Linux 2.6.29
	I/O Scheduler	NOOP
	Android Version	2.3
Simulation Environment	OS Kernel	Linux 3.0
	Flash Size	1GB
	PCM Size	128Mb

TABLE II: Trace Applications

Trace Name	Applications
NetApps	Web Browser, EMail, Search, Settings
MultiApps	Music, Camera, Gallery, Settings
CommApps	Phone, Contacts, Messaging, Voice Dialer
MixedApps	Browser, EMail, Music, Contacts, Settings

B. Results and Discussion

Experimental results in Table III show that WAB-FTL can significantly reduce write activities in PCM in comparison with BL-FTL. Compared with BL-FTL, WAB-FTL can achieve an average reduction of 80.76% and a maximum reduction of 82.61% in the maximum number of bit flips. As shown, WAB-FTL reduces almost half of write activities in PCM. For the total number of bit flips, WAB-FTL reduces 52.09% bit flips on average, with a maximum reduction of 57.77%. Therefore, WAB-FTL is more applicable in PCM-based embedded systems. By adopting WAB-FTL, the lifetime of PCM can be prolonged.

TABLE III: The Comparison of PCM Bit Flips

Trace	Total Number of Bit Flips			Maximum Number of Bit Flips		
	BL-FTL	WAB-FTL	WAB-FTL over BL-FTL	BL-FTL	WAB-FTL	WAB-FTL over BL-FTL
NetApps	98582253	44842764	54.51%	170523	32948	80.68%
MultiApps	86669414	50804270	41.38%	118046	25110	78.73%
CommApps	101417631	45938112	54.70%	171381	32527	81.02%
MixedApps	66624451	28135697	57.77%	129795	22571	82.61%
Avg			52.09%			80.76%

V. CONCLUSION

In this paper, we have proposed a write-activity-aware block-level flash memory management scheme, WAB-FTL, which can effectively reduce write activities in PCM-based embedded systems. In WAB-FTL, the performance improvement is achieved by preserving a bit in a PCM cell from being inverted frequently. Through the proposed Lazy-Merge strategy and Cooling-Pool in PCM, unnecessary write activities to PCM-based embedded systems are directly reduced. We conducted experiments on a set of realistic I/O traces collected from Google Android™2.3. Experimental results demonstrate the advantage of WAB-FTL in write activity reduction when compared with the baseline scheme.

ACKNOWLEDGMENT

The work described in this paper is partially supported by the grants from the Innovation and Technology Support Programme of Innovation and Technology Fund of the Hong Kong Special Administrative Region, China (ITS/082/10).

REFERENCES

[1] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Design Test of Computers*, vol. 28, no. 1, pp. 44–51, Jan.–Feb. 2011.

[2] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proceedings of MICRO '09*, 2009, pp. 14–23.

[3] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing PCM main memory lifetime," in *Proceedings of DATE '10*, 2010, pp. 914–919.

[4] International Technology Roadmap for Semiconductors, "Process itegration, devices, and structures (2007 edition)," *http://developer.intel.com*, 2007.

[5] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng, "A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems," in *Proceedings of EMSOFT '08*, 2008, pp. 31–40.

[6] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *Proceedings of HPCA '10*, Jan. 2010, pp. 1–12.

[7] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of ISCA '09*, 2009, pp. 14–23.

[8] G. Sun, D. Niu, J. Ouyang, and Y. Xie, "A frequent-value based PRAM memory architecture," in *Proceedings of ASP-DAC '11*, 2011, pp. 211–216.

[9] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H.-M. Sha, "Reducing write activities on non-volatile memories in embedded CMPs via data migration and recomputation," in *Proceedings of DAC '10*, 2010, pp. 350–355.

[10] T. Wang, D. Liu, Z. Shao, and C. Yang, "Write-activity-aware page table management for PCM-based embedded systems," in *Proceedings of ASP-DAC '12*, 2012.

[11] Intel Corporation., "Understanding the flash translation layer (FTL) specification," *http://developer.intel.com*, 2009.

[12] T.-W. Kuo, Y.-H. Chang, P.-C. Huang, and C.-W. Chang, "Special issues in flash," in *Proceedings of ICCAD '08*, November 2008, pp. 821–826.

[13] A. Ban, "Flash file system optimized for page-mode flash technologies," *US patent 5,937,425*, August 1999.

[14] C.-H. Wu and T.-W. Kuo, "An adaptive two-level management for the flash translation layer in embedded systems," in *Proceedings of ICCAD '06*, 2006, pp. 601–606.

[15] Y. Wang, D. Liu, Z. Qin, Z. Shao, and Y. Guan, "RNFTL: a reuse-aware NAND flash translation layer for flash memory," in *Proceedings of LCTES '10*, vol. 45, no. 4, 2010, pp. 163–172.

[16] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao, "PCM-FTL: a write-activity-aware NAND flash memory management scheme for PCM-based embedded systems," in *Proceedings of RTSS '11*, 2011, pp. 357–366.

[17] Micron Technology, Inc.., "Micron phase change memory," *http://www.micron.com/products/pcm/*, 2011.