

# Chapter 6

## NVRAM-Assisted Optimization Techniques for Flash Memory Management in Embedded Sensor Nodes

Duo Liu and Kan Zhong

**Abstract** Embedded sensor nodes are sensitized to battery lifetime and evidences show that DRAM-based main memory subsystem is the major contributor of the energy consumption of embedded sensor nodes. Due to the high density, byte-addressability, and low standby power consumption, non-volatile random access memories (NVRAMs), such as PRAM and STT-RAM, become promising main memory alternatives in embedded sensor nodes. On the other hand, NAND flash memory is widely adopted for storing collected data in embedded sensor nodes. However, both NVRAM and NAND flash memory have limited lifetime, how to optimize the management of NAND flash memory in NVRAM-based embedded sensor nodes while considering the endurance issue becomes quite important. In this chapter, we introduce a write-actively-aware NAND flash memory management scheme to effectively manage NAND flash memory while reducing the write activities to NVRAM-based main memory in embedded sensor nodes. The basic idea is to preserve each bit in flash mapping table, which is stored in NVRAM, from being inverted frequently during the mapping table update process. To achieve this, a two-level mapping mechanism is employed while considering the access behavior of IO requests, and a customized wear-leveling scheme is developed to evenly distribute the writes across the whole mapping table. Evaluation results show that the proposed technique can reduce the write activities significantly and achieve an even distribution of writes in NVRAM with low overhead.

**Keywords** Sensor node • Embedded systems • Flash translation layer • Flash memory • Non-volatile memory • Phase change memory • NVRAM • Wear-leveling • Endurance

---

D. Liu (✉) • K. Zhong  
College of Computer Science, Chongqing University, No. 174 Shazhengjie,  
Shapingba, Chongqing 400044, China  
e-mail: [liuduo@cqu.edu.cn](mailto:liuduo@cqu.edu.cn); [kzhong1991@cqu.edu.cn](mailto:kzhong1991@cqu.edu.cn)

## 6.1 Introduction

Due to the limitation in size and cost, embedded sensor nodes are commonly equipped with a small battery, which has limited capacity, making the sensor nodes sensitive to battery lifetime. Recent researches report that DRAM-based main memory subsystem has become the major contributor of the embedded system's overall energy consumption [1, 2]. To solve this problem, a mount of researches argue that non-volatile random access memory (NVRAM), such as phase change memory (PCM) [3–10], spin-transfer torque random access memory (STT-RAM) [11–13], is a promising DRAM alternative [14]. However, compared to DRAM, NVRAM exhibits limited endurance (e.g.,  $10^6$ – $10^8$  for PRAM cells) and high write latency/energy [15]. These constraints impose challenges for using NVRAM as a complete replacement for DRAM. On the other hand, NAND flash memory has been widely used in embedded sensor nodes due to its attractive features, such as shock resistance, low power, and high density [16]. To manage flash memory, FTL is introduced to emulate NAND flash memory as a block device interface for file systems [17]. The FTL functions as translating logical addresses of I/O requests into physical addresses in NAND flash memory. To achieve this, FTL maintains a mapping table (i.e., metadata), which stores the mapping information between logical addresses and physical addresses. The mapping table is usually cached in main memory for better performance and written back to NAND flash periodically.

Over the past decade, many studies for FTL schemes have been proposed [18–27]. According to the granularity of mapping unit, there are three types of FTL schemes: page-level mapping, block-level mapping, and hybrid-level mapping [17]. Most of the previous work, however, have not yet explored the management mechanism of NAND flash memory in the emerging NVRAM-based embedded sensor nodes. Kim et al. [28] propose a page-level mapping FTL (*hFTL*) for managing NAND flash memory in the NVRAM-based embedded systems, where the page-level mapping table is stored in NVRAM and user data is stored in NAND flash memory. Nevertheless, their approach does not consider write activities of FTL mapping table in NVRAM, and the access behavior of I/O requests as well. As FTL mapping table is updated frequently in NVRAM, a huge number of unnecessary write operations on FTL mapping table will degrade the endurance of NVRAM. New techniques, therefore, are needed to eliminate unnecessary write operations on FTL mapping table and, at the same time, to enhance the endurance of NVRAM-based sensor nodes.

In this chapter, we introduce a write-activity-aware two-level FTL scheme, called **NV-FTL**, to effectively manage NAND flash memory and enhance the endurance of NVRAM-based embedded sensor nodes. Different from existing approaches [29–34], NV-FTL enhances the lifetime of NVRAM by making the management of NAND flash memory aware of write activities on underlying memory architecture. With NV-FTL, no change to the file systems or hardware implementation of NAND flash and NVRAM is required. Our basic idea is to preserve each bit in FTL mapping table, which is stored in NVRAM, from being

inverted frequently, i.e., we focus on minimizing the number of bit flips in an NVRAM cell when updating the FTL mapping table. NV-FTL employs a two-level mapping mechanism, which not only focuses on minimizing write activities of NVRAM but also considers the access behavior of I/O requests. To achieve this, in NVRAM, we use a page-level mapping table to handle not frequently updated random requests, and allocate a tiny buffer of block-level mapping table to record most frequently updated sequential requests. To further minimize write activities in NVRAM, NV-FTL actively chooses a physical block in NAND flash memory whose physical block number (PBN) incurs minimum number of bit flips. Consequently, the write activities are eliminated and the endurance of NVRAM is enhanced.

We conduct trace-driven experiments with both general purpose and mobile I/O workloads to show the effectiveness and versatility of NV-FTL. A representative FTL design *h*FTL [28] for NVRAM-based embedded systems is selected as a baseline scheme. The proposed NV-FTL is compared with *h*FTL in terms of NVRAM bit flips with various configurations. The experimental results show that our approach can achieve an average reduction of 93.10% and a maximum reduction of 98.98% in the maximum number of bit flips for an NVRAM-based embedded sensor nodes with 1 GB NAND flash memory. In addition, the results also show that NV-FTL can achieve an even distribution of bit flips in NVRAM when compared with the baseline scheme.

The rest of this chapter is organized as follows. Section 6.2 introduces the background and motivation. Section 6.3 presents our proposed NV-FTL technique. Section 6.4 reports the experimental results. Finally, in Sect. 6.5, we present the conclusion.

## 6.2 Background and Motivation

In this section, we first introduce the background knowledge of NVRAM-based embedded sensor node. Then we describe the issues of a representative FTL scheme. Finally, we present the motivation of our work.

### 6.2.1 NVRAM-Based Sensor Node

Figure 6.1 shows a typical NVRAM-based embedded sensor node. As shown, NVRAM is served as the sensor node's main memory and NAND flash memory is adopted as the storage media. Analog signals collected by various sensors are firstly convert into digital signals by the analog digital converter (ADC), and then the digital signals are processed by the CPU and stored in the storage system [35], in which the FTL mapping table is cached in NVRAM and sensor data are stored in NAND flash memory. In the storage system, the MTD layer provides primitive functions such as read, write, and erase operations. The FTL layer emulates the

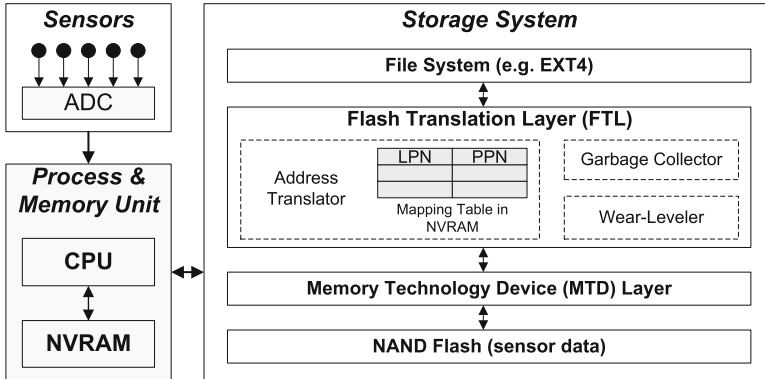


Fig. 6.1 Illustration of NVRAM-based embedded sensor node with NAND flash memory

flash memory as a disk device so that it can provide transparent storage service to file systems. Following the I/O requests, FTL translates addresses between logical page number (LPN) and physical page number (PPN), and keeps track of the mapping information by using an FTL mapping table in NVRAM. Then according to the mapping, data can be directly read from (write into) NAND flash memory.

Unlike NAND flash memory, NVRAMs support bit-addressability and in-place update. NVRAMs keep data by changing the physical state of its underlying material without maintaining constant current. One promising candidate is PCM, which stores data by changing the state of the phase change material (e.g., GST). By ejecting electrical pulses to heat up the GST region, each PCM cell can switch between two states—amorphous and crystalline, which have high and low electrical resistance, respectively. Reading a bit from a PCM cell is accomplished by sensing the resistance level of the cell. To represent binary “1”, a SET operation is performed to turn a PCM cell into the crystalline state by applying a moderate power, long duration pulses; To represent binary “0”, a RESET operation is performed to turn a PCM cell into the amorphous state by applying a high power, short duration pulses. Both of these operations impose heat stress to PCM cells, and thus a PCM cell can only sustain a limited number of write (SET/RESET) operations (e.g.,  $10^6$ – $10^8$  for Micron P5Q PCM [36]). Other NVRAM like STT-RAM, memristor [37] also suffers from the endurance problem. In this chapter, we do not target at any specific NVRAM, we target at the optimization of NAND flash memory management in NVRAM-based sensor nodes since all the NVRAMs have the same problem—limited endurance.

## 6.2.2 A Representative FTL Scheme

In this section, we briefly revisit the *h*FTL scheme which is proposed for managing NAND flash memory in PCM-based embedded systems [28].

*h*FTL is based on page-level mapping scheme [18], but it is optimized for PCM-based embedded systems. *h*FTL stores metadata such as FTL mapping

table, physical page information, and physical block information in PCM. NAND flash memory is only used for storing user data from the file system, and the blocks in NAND flash memory are categorized into three types, i.e., garbage blocks, data blocks, and a buffer block. Different from the conventional page-level mapping FTL, *hFTL* uses a buffer block to store the newly arrived data. When the buffer block runs out of free pages, it is put into the data block list and another empty buffer block is allocated from the garbage block list. If there is not enough number of garbage blocks, a garbage collection operation is performed to reclaim a block from the data blocks. In *hFTL*, a page-level mapping table in PCM keeps track of mappings between LPN and PPN, in terms of the I/O requests. Consequently, the mapping table is updated frequently and thus imposes the endurance issue for PCM. A motivational example is illustrated in Fig. 6.2.

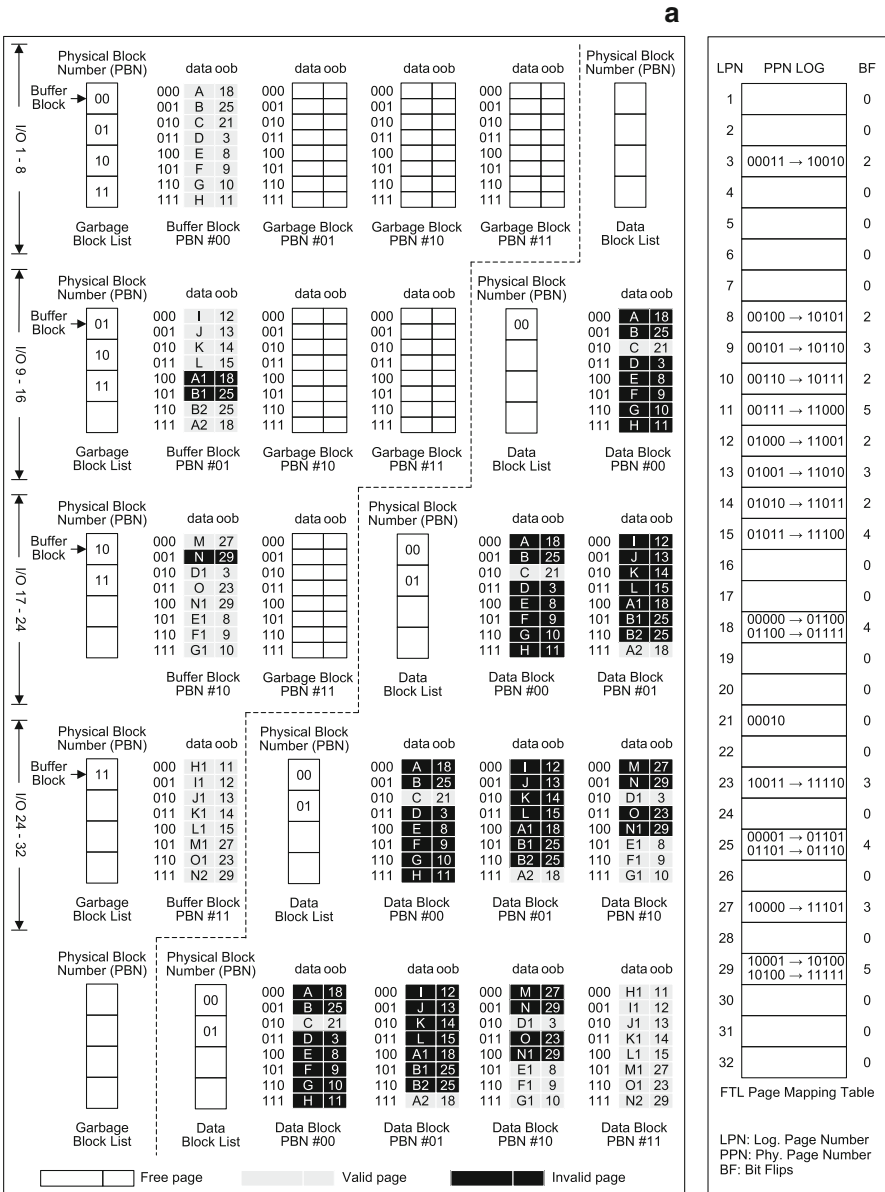
In the example, we presume that PCM is adopted as the main memory of sensor node, and there are four blocks in NAND flash memory, and each block has 8 pages. Therefore, a page-level mapping table in PCM has 32 entries to record the mapping information. To facilitate the comparison of *hFTL* and our NV-FTL scheme, the PPN, PBN, and the offset of each block are represented by binary number. We assume that each entry of the mapping table is empty at the beginning, and the binary number in an entry is the updated PPNs to reflect the updates of mapping. The I/O access requests of write operations (w) are listed in Fig. 6.2a. According to the given I/O requests, the status variation of the blocks in NAND flash memory is shown in Fig. 6.2b. For *hFTL*, when a write operation is performed, the corresponding content is first written to a free page of the current buffer block in a sequence order.

As shown, the first request is written to LPN (#18). A new buffer block (PBN #00) is allocated from the garbage block list, and the content *A* with the corresponding LPN (#18) is stored in the first page of current buffer block (PBN #00). Meanwhile, the mapping information of LPN (#18) and PPN (#00000) is stored into the mapping table shown in Fig. 6.2c. Note that PPN is the combination of PBN and the block offset. After serving the eighth request, buffer block (PBN #00) is full and becomes a data block. Likewise, the remaining garbage blocks (PBN #01, PBN #10, and PBN #11) are allocated as a buffer block, respectively, to serve the following write operations. Finally, when the content of *N2* with the corresponding LPN (#29) is written into the last page of buffer block (PBN #11), all garbage blocks become data blocks and some entries of the mapping table have been updated by new PPNs for several times.

### 6.2.3 Motivation

In the motivational example, several update operations are performed in the FTL page-level mapping table. For instance, the 13th request updates the old content in the 1st page of data block (PBN #00) by setting that page invalid, and writes the new content to the current buffer block (PBN #01). Meanwhile, the corresponding

I/O Requests	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
Command	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
Logical Page Number (LPN)	18	25	21	3	8	9	10	11	12	13	14	15	18	25	18	27	29	3	23	29	8	9	10	11	12	13	14	15	27	23	29		
Content	A	B	C	D	E	F	G	H	I	J	K	L	A1	B1	B2	A2	M	N	D1	O	N1	E1	F1	G1	H1	I1	J1	K1	L1	M1	O1	N2	



**Fig. 6.2** Motivational example. (a) I/O access requests. (b) The status variation of blocks in NAND flash memory. (c) The status variation of FTL page-level mapping table in PCM

mapping information in the mapping table is updated as well. In Fig. 6.2c, we use the bit flips (BF), shown on the right side of the mapping table, to reflect the update frequency of each entry in the mapping table. As shown, the 11th and 29th entry have the maximum number of bit flips 5. Since NVRAM cell, like PCM, can only sustain limited number of write cycles, frequent update operations in mapping table will lead to the fast worn out of NVRAM. These observations motivate us to propose a write-activity-aware FTL to effectively manage NAND flash memory and, at the same time, to improve the endurance of NVRAM-based embedded sensor node.

As mentioned above, several hardware optimization techniques for NVRAM have been proposed [38–40] to tackle the redundant write activities by eliminating a write if its designated memory cell holds the same value. Then through utilizing such a fine-grained hardware feature, this work actively chooses mapping information (e.g., PBN) which is almost the same as the mapping to be updated in the mapping table, such that the number of write activities in NVRAM is minimized.

### 6.3 NV-FTL: Write-Activity-Aware FTL

In this section, we present the details of our NV-FTL, a write-activity-aware FTL, that can effectively enhance the endurance of the NVRAM-based embedded sensor node. We first present an overview of NV-FTL in Sect. 6.3.1. We then provide a detailed description of NV-FTL in Sect. 6.3.2.

#### 6.3.1 Overview

The objective of NV-FTL is to reduce write activities in NVRAM-based embedded sensor node, and therefore, the endurance of NVRAM is enhanced. So the basic idea of NV-FTL is to preserve each bit in FTL mapping table, which is stored in NVRAM, from being inverted frequently, i.e., we focus on minimizing the number of bit flips in an NVRAM cell when updating the FTL mapping table. Different from the previous work [28], our NV-FTL adopts a two-level mapping mechanism, which not only focuses on minimizing write activities in NVRAM but also considers the access behavior of I/O requests. NV-FTL uses a page-level mapping table to record the mapping of write requests not frequently updated, and allocates a tiny buffer of block-level mapping table to cache the mapping of those most frequently updated write requests. With the consideration of write activities, once a block is needed for incoming write requests, NV-FTL actively chooses a physical block in NAND flash memory whose PBN incurs minimum number of bit flips.

By applying NV-FTL, the number of bit flips is reduced, and thus the number of write activities in NVRAM is minimized. Consequently, the endurance of the NVRAM-based embedded sensor node is enhanced.

### 6.3.2 NV-FTL Description

In general, a realistic I/O workload is a mixture of random and sequential requests. By separating the random requests from the sequential requests, we can not only obtain the access behavior but also handle those frequently updated write requests. Otherwise, without considering the access behavior of I/O workload, we cannot effectively manage NAND flash memory and may waste lots of blocks in garbage collection due to frequent update operations. Therefore, in NV-FTL, we design a behavior detector to separate the I/O workload into random and sequential requests, according to the length of each request in the I/O workload. The length is a user-defined threshold, which is determined by observing performance gains with different threshold values (e.g., 8, 16, and 32) in the experiments. For example, if the length of a request is smaller than 8, then this request is treated as a random request; Otherwise, if the length of a request is greater than or equal to 8, then it is treated as a sequential request.

Figure 6.3 shows the structure of NV-FTL. As shown, NV-FTL first separates the I/O workload into random requests and sequential requests. Then NV-FTL adopts a two-level FTL mechanism to handle these two cases as follows:

- For random requests: NV-FTL sequentially allocates physical pages from the first page of a physical block in NAND flash memory, so that all pages in blocks are fully utilized. Accordingly, NV-FTL adds LPN to PPN mapping of random requests into the page-level mapping table.
- For sequential requests: NV-FTL allocates physical pages based on block offset as most sequential requests usually occupy a whole block, so that all pages in blocks are fully utilized as well. Similarly, NV-FTL adds an LBN to PBN mapping of sequential requests into the block-level mapping table buffer.

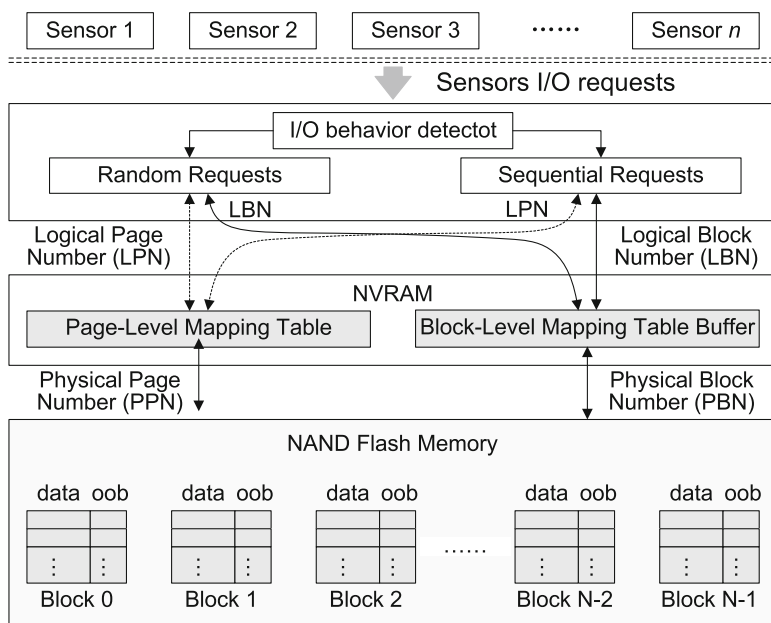


Fig. 6.3 Structure of NV-FTL



In NV-FTL, we only allocate a tiny buffer for temporary storing a part of the block-level mapping table. For example, the size of this block-level mapping buffer is set as 10% of the size of the original block-level mapping table. Therefore, a replacement policy should be considered when the buffer is full. Similar as a cache, we only kick out the mapping of those not frequently updated blocks, while maintaining the mapping of frequent updated blocks. The kicked out mapping information is put into the page-level mapping table. If a block in NAND flash memory has  $N_p$  valid pages, and its corresponding block-level mapping is kicked out to page-level mapping table, then  $N_p$  entries in page-level mapping table should be filled with the corresponding LPN to PPN mapping for each page in the block. On the contrary, the page-level mapping of a block can be re-added into the block-level mapping table buffer, once the block is updated again by sequential write requests. Therefore, by observing the frequently updated requests, our technique can dynamically adjust the block-level mapping table buffer and the page-level mapping table, such that write activities of frequently updated requests are only buffered in block-level mapping table buffer which only contributes a small number of bit flips in NVRAM. The experimental results in Sect. 6.4 confirms this fact.

To further minimize write activities in NVRAM, a write-activity-aware strategy is proposed. In our technique, to allocate a new block for the write/update requests, the corresponding original physical block number (PBN) is first obtained from page-level mapping table (by dividing PPN with the number of pages in a block), or from block-level mapping table buffer with the requested LPN. Then according to the original PBN, we actively select a physical block in NAND flash memory whose PBN is almost the same as the original PBN, i.e., the new PBN incurs minimum number of bit flips if the original PBN is updated by the new PBN in the mapping table. As a result, a large number of redundant bit flips are reduced, and the endurance of NVRAM is enhanced.

---

### Algorithm 6.3.1 The algorithm of NV-FTL

---

**Require:** I/O requests with random request or/and sequential request.

**Ensure:** Allocate pages for the I/O request.

- 1: Divide the I/O request into random writes or/and sequential writes according to a predefined threshold.
- 2: **if** Random write request arrives **then**
- 3: Obtain the *LBN* and *LPN* of the random write request.
- 4: **if** *LBN*'s mapping is not in block-level mapping table buffer or *LPN*'s mapping is not in page-level mapping table **then**
- 5: This is a new write, allocate a new block *PBN*, and write the contents into the block sequentially from the first page.
- 6: Add the mapping of (*LPN*, *PPN*) into the page-level mapping table.
- 7: **end if**

---

(continued)

---

**Algorithm 6.3.1** (continued)
 

---

```

8:  if LBN's mapping exists in block-level mapping table buffer or LPN's
    mapping exists in page-level mapping table then
9:    This is an update, obtain the PBN of the updated block.
10:   if There exists enough space in the PBN block for the update
    request then
11:     Write the update contents in the left space of the PBN block sequen-
    tially, and invalidate the old pages in the same block.
12:   else
13:     Actively find a new block whose block number is almost the same as
    PBN, write the update contents in the new block sequentially, and
    invalidate the old pages in PBN block.
14:   end if
15:   Update block-level mapping table buffer or page-level mapping table.
16: end if
17: end if
18: if Sequential write request arrives then
19:   Obtain the LBN and LPN of the sequential write request.
20:   if LBN's mapping is not in block-level mapping table buffer or LPN's
    mapping is not in page-level mapping table then
21:     This is a new write, allocate a new block PBN, and write the contents of
    the request into the block based on block offset.
22:     if The block-level mapping table buffer is full then
23:       Kick out least frequently used entry, add the kicked out mappings into
       page-level mapping table.
24:     end if
25:     Add the mapping of (LBN, PBN) into the block-level mapping table
       buffer.
26:   end if
27:   if LBN's mapping exists in block-level mapping table buffer or LPN's
    mapping exists in page-level mapping table then
28:     This is an update, obtain the PBN of the updated block.
29:     if There exists enough space in the PBN block for the update
    request then
30:       Write the update contents in the left space of the PBN block based on
       block offset, and invalidate the old pages in the same block.
31:     else
32:       Actively find a new block whose block number is almost the same as
       PBN, write the update contents in the new block based on block offset,
       and invalidate the old pages in PBN block.
33:     end if
34:     Update block-level mapping table buffer or page-level mapping table.
35:   end if
36: end if

```

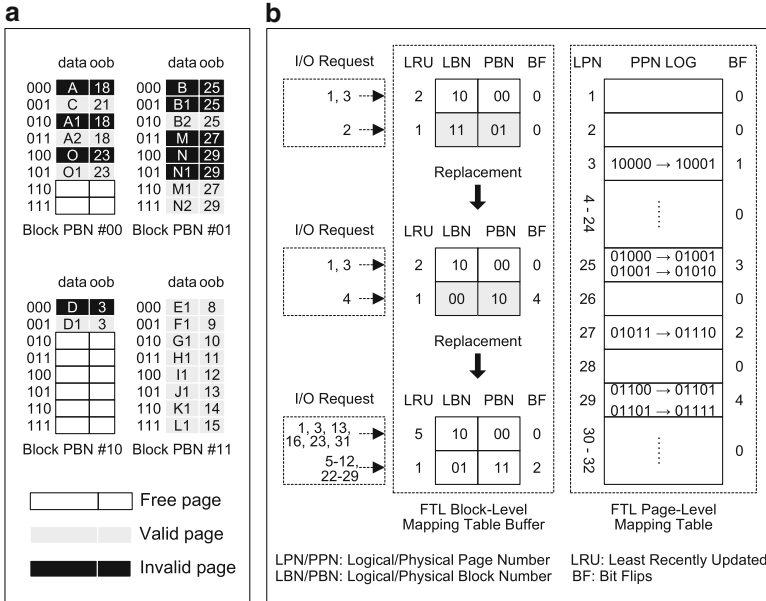
---

Algorithm 6.3.1 shows the process of a write operation of NV-FTL. NV-FTL first divides the incoming I/O request into random writes or/and sequential writes according to a threshold. Then the random and sequential write requests are processed separately. For random write request (lines 2–17), if it is a new write, i.e., we cannot find its corresponding *LBN* or *LPN* mapping in the block-level mapping table buffer or page-level mapping table. So NV-FTL finds a new block *PBN*, and write the contents of the random write request into the allocated new block sequentially from the first page. After that, we add the (*LPN*, *PPN*) mapping into the page-level mapping table. If the random write request is an update, and there exists enough space in the updated block, then write the update contents into the left space of the block sequentially, and invalid the old pages in the same block. Otherwise, there does not exist enough space in the updated block, NV-FTL will actively find a new block whose block number is almost the same as *PBN*, and then write the update contents in the new block based on block offset. At last, we update the corresponding block-level mapping table buffer or page-level mapping table. For sequential write request (lines 18–36), we process it in the similar way as that for processing random write request.

Note that the block-level mapping table buffer is updated frequently by sequential write requests, so it may become very hot and lead to an uneven distribution of bit flips in NVRAM. To avoid this scenario and enhance NVRAM endurance, a wear-leveling method is integrated into NV-FTL. In NV-FTL, during a period of time (e.g., every 100 I/O requests), the block-level mapping table buffer is moved across the whole mapping table area (block-level and page-level mapping table) in NVRAM. With acceptable copy operations of mapping information, an even distribution of bit flips in NVRAM is obtained.

An example of NV-FTL is shown in Fig. 6.4. This example is based on the I/O requests and the NAND flash memory assumptions for the motivational example shown in Fig. 6.2. As shown, for the first random request with *LPN* (#18), we find a new block (*PBN* #00), and the content *A* is written sequentially into the first page (#00000) of block (*PBN* #00). For this request, there is no bit flip when updating the mapping table. It can be seen that *A* is updated by a new content *A1* in the 13th request, and *A1* is written into the physical page (#00010) according to the update policy of NV-FTL. When the 13th request arrives, we use the *LPN* (#18) to get the corresponding *LBN* (#10). Then we find the *LBN* (#10) is already in the block-level mapping table buffer, so the 13th request is an update to the old page in the block (*PBN* #00), then by checking the block (*PBN* #00), we know the old content *A* of this *LPN* (#18) is stored in the page *PPN* (#00000), thus this page is set as invalid. Since there exists enough space in block (*PBN* #00), the new update content *A1* of *LPN* (#18) is written sequentially into the block.

It is noticed that the 5th to 12th requests form a sequential write, then we allocate a new block (*PBN* #11) for this request, and write the contents into each page of the block based on offset. The corresponding *LBN* to *PBN* mapping (01, 11) is added into the block-level mapping table buffer. Later, when the following 22nd to 29th sequential update requests arrive, then the old pages in the block (*PBN* #11) are invalid. Since we cannot find free block, the block (*PBN* #11) is erased, and the new



**Fig. 6.4** Illustration of NV-FTL. (a) The status variation of blocks in NAND flash memory according to the access sequence in Fig. 6.2. (b) The status variation of FTL page-level mapping table and block-level mapping table buffer in NVRAM

update data E1 to L1 is written into this block based on offset. Finally, we update the block-level mapping table buffer, and the value of corresponding LRU is updated as well.

After processing all requests, we found that the total number of bit flips in NVRAM is 16 by our NV-FTL, while the total number of bit flips in NVRAM are 44 by *h*FTL. Our scheme achieves a reduction of 63.6% in the total number of bit flips, which confirms that our approach can effectively reduce write activities in NVRAM. The experimental results in Sect. 6.4 also show that our scheme can effectively reduce the total number of bit flips.

## 6.4 Evaluation

To evaluate the effectiveness of the proposed NV-FTL, we conduct a series of experiments and present the experimental results with analysis in this section. We compare and evaluate our proposed NV-FTL scheme over the representative page-level FTL scheme, *h*FTL[28].

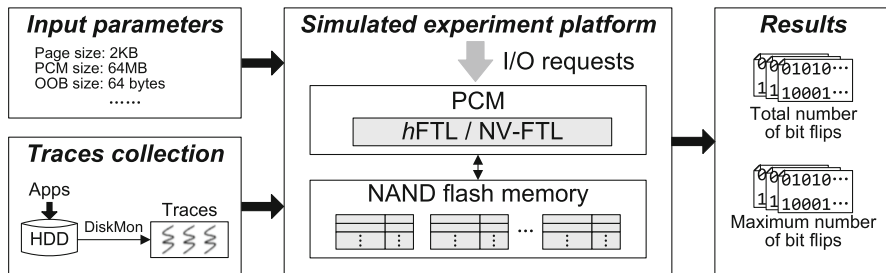


Fig. 6.5 The framework of simulation platform

### 6.4.1 Experimental Setup

Although our design does not target at any specific NVRAM, in the evaluation, we assume that a particular NVRAM—PCM—is used as the main memory of embedded sensor node. The evaluation is through a trace-driven simulation. The framework of our simulation platform is shown in Fig. 6.5. In our experiment, we use the same experimental configuration adopted by hFTL [28], a 1 GB NAND flash memory and a 64 MB PCM are configured in our simulator. The traces along with various flash parameters, such as block size and page size, are fed into our simulation framework. The page size, number of pages in a block, and size of the OOB for each page are set as 2 KB, 64, and 64 Bytes, respectively. Therefore, the 1 GB NAND flash memory used in the experiment has 8,192 physical blocks. To fully evaluate our scheme, we further conduct the experiments on a 4 GB NAND flash memory with the same configurations. In addition, the threshold for distinguishing random and sequential requests is set as 8.

To represent the realistic I/O request patterns, we collected the traces from desktop running DiskMon [41] with an Intel Pentium Dual Core 2 GHz processor, a 200 GB hard disk, and a 2 GB DRAM. Among these traces, CopyFiles is a trace collected by copying files from hard disk to an external hard drive; DownFiles represents a trace collected by downloading files from a network server; Office represents a trace collected by running some office related applications; P2P represents a trace collected by running a P2P file-sharing application on an external hard drive; Table 6.1 summarizes our experimental platform and trace collection environment.

### 6.4.2 Results and Discussion

In this section, we present the experimental results with analysis. We first present the endurance impact of NV-FTL. Then we present the wear-leveling comparison of NV-FTL and the baseline scheme.

**Table 6.1** Experimental setup

Hardware	CPU	Intel dual core 2 GHz
	Disk space	200 GB
	RAM	2 GB
Simulation environment	OS kernel	Linux 2.6.17
	Flash size	1 GB & 4 GB
	PCM	64 MB
Trace	OS	Windows XP (NTFS)
	Trace name generator	DiskMon
	Applications	Web applications, MSN, Word, Excel, PowerPoint, Media player, Emuler

#### 6.4.2.1 NVRAM Endurance

The objective of this work is to reduce write activities to enhance the endurance of NVRAM-based embedded sensor node. Therefore, the endurance of NVRAM is one of the most important factors in analyzing the reliability of NVRAM-based embedded sensor node. The endurance of NVRAM is mainly affected by the worst case of bit flips in an NVRAM cell, i.e., the maximum number of bit flips in a NVRAM cell determines the endurance of NVRAM. For example, if PCM can only sustain  $10^6$  write cycles, then a PCM cell is worn out if it suffers from more than  $10^6$  bit flips. So our technique not only focuses on minimizing write activities in NVRAM but also reducing the maximum number of bit flips. Table 6.2 presents the results for the maximum and total number of bit flips among all PCM cells when managing 1 GB and 4 GB NAND flash memory embedded sensor node.

We observe that NV-FTL can significantly reduce write activities of PCM in comparison with the baseline scheme—*h*FTL. As shown in the table, for the embedded sensor node with 1 GB NAND flash memory, NV-FTL can achieve more than 60 % reduction of total number of bit flips. Similarly, for 4 GB NAND flash memory, NV-FTL also achieves a great amount of total number of bit flips when compared with the baseline scheme, proving that NV-FTL can effectively preserve the PCM cells being converted frequently.

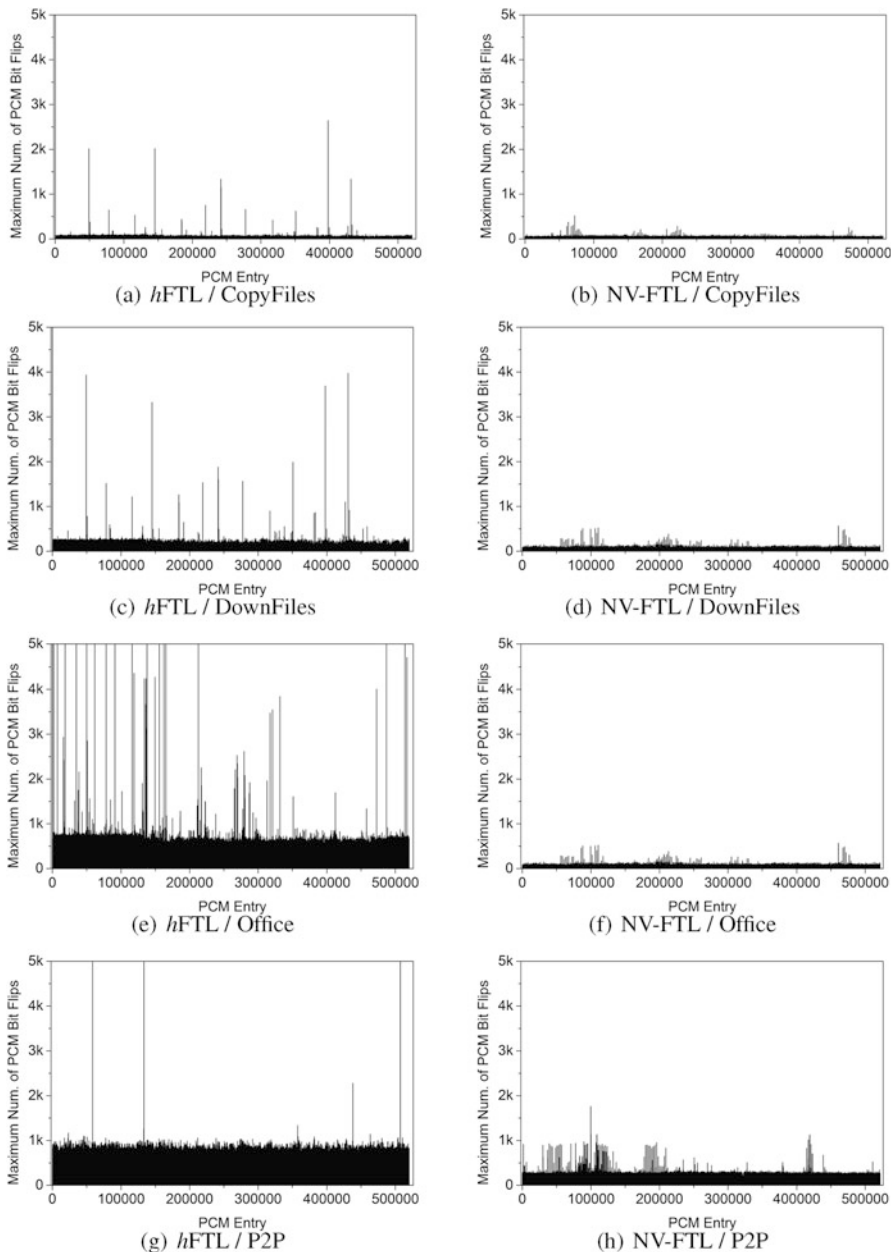
Moreover, in terms of maximum of bit flips, NV-FTL exhibits a similar trend in the result of total number of bit flips. When compared with the baseline scheme, NV-FTL can reduce the maximum number of bit flips more than 90 and 80 % for 1 GB and 4 GB NAND flash memory, respectively. To some extent, the reduction in maximum number of bit flips can slow down the wearing out of certain NVRAM cells. Since NV-FTL can reduce both the total number of bit flips and maximum number of bit flips, we therefore conclude that NV-FTL can effectively prolong the endurance of NVRAM, such as PCM, making the NVRAM-based embedded sensor node has a longer longevity.

**Table 6.2** The maximum and total number of bit flips of NV-FTL versus hFTL

<i>PCM with 1 GB NAND flash memory</i>					
Trace name	% of write	% of read	Total number of bit flips		
			hFTL	NV-FTL	NV-FTL over hFTL (%)
CopyFiles	78.75	21.25	559,496,658	293,866,292	47.48
DownFiles	71.88	28.12	1,756,464,372	568,987,257	67.61
Office	77.37	22.63	7,520,028,995	2,576,892,175	65.73
P2P	28.95	71.05	6,929,967,624	1,718,812,456	75.20
Average					64.00
Trace name	% of write	% of read	Maximum number of bit flips		
			hFTL	NV-FTL	NV-FTL over hFTL (%)
CopyFiles	78.75	21.25	9,977	519	94.80
DownFiles	71.88	28.12	21,945	567	97.42
Office	77.37	22.63	9,385	1,762	81.23
P2P	28.95	71.05	74,540	762	98.98
Average					93.10
<i>PCM with 4 GB NAND flash memory</i>					
Trace name	% of write	% of read	Total number of bit flips		
			hFTL	NV-FTL	NV-FTL over hFTL (%)
CopyFiles	78.75	21.25	122,605,530	93,456,325	23.77
DownFiles	71.88	28.12	842,401,436	191,579,924	77.26
Office	77.37	22.63	6,981,790,260	919,567,590	86.83
P2P	28.95	71.05	15,269,865,958	650,611,634	95.74
Average					70.90
Trace name	% of write	% of read	Maximum number of bit flips		
			hFTL	NV-FTL	NV-FTL over hFTL(%)
CopyFiles	78.75	21.25	10,461	2,076	80.15
DownFiles	71.88	28.12	20,857	3,923	81.19
Office	77.37	22.63	36,667	7,377	79.88
P2P	28.95	71.05	86,383	7,623	91.18
Average					83.10

#### 6.4.2.2 PCM Wear-Leveling

Wear-leveling is another major concern in NVRAM-based embedded sensor node. A good wear-leveling not only prolong NVRAM-based embedded sensor node's longevity, but also increase its reliability since worn out cells may lead to corrupted data. Therefore, in Fig. 6.6, we plot the maximum number of bit flips among all mapping table entries for sensor node with 64 MB PCM and 1 GB NAND flash memory. For each subfigure, the  $x$ -axis denotes the number of page-level and block-level mapping table entries in PCM, while the  $y$ -axis shows the maximum number



**Fig. 6.6** The wear-leveling comparison of *hFTL* and NV-FTL in a PCM-based embedded sensor node with 1GB NAND flash memory over four traces. (a) *hFTL*/CopyFiles, (b) NV-FTL/CopyFiles, (c) *hFTL*/DownFiles, (d) NV-FTL/DownFiles, (e) *hFTL*/Office, (f) NV-FTL/Office, (g) *hFTL*/P2P, (h) NV-FTL/P2P



of bit flips extracted from each mapping table entry. To present the distributions clearly, we restrict the maximum number of bit flips on y-axis to 5,000.

As shown in the figure, we observe that the write distribution of bit flips for *hFTL* varies a lot, and this will surely pose a threat to the endurance of PCM, making certain PCM cells worn out quickly, as well as other NVRAMs. However, compared to *hFTL*, NV-FTL distributes write activities more evenly among all PCM cells, especially for DownFiles and Office, the wear-leveling of which have a great improvements in NV-FTL. The results listed in Table 6.2 also illustrate this fact. In summary, NV-FTL achieves much better wear-leveling than the baseline scheme, leading to the NVRAM-based embedded sensor nodes have a better reliability.

## 6.5 Conclusion

In this chapter, we have proposed a write-activity-aware NAND flash memory management scheme NV-FTL which takes the first step to reduce write activities in NVRAM-based sensor node. In our NV-FTL, the performance improvement is achieved by preserving a bit in an NVRAM cell from being inverted frequently. Through a two-level mapping mechanism, and a write-activity-aware strategy, unnecessary write activities in NVRAM are directly eliminated. We conducted experiments on a set of realistic I/O workload collected from daily-life. For a sensor node with 64 MB PCM and 1 GB (4 GB) NAND flash memory, the experimental results show that the maximum number of bit flips among PCM cells can be reduced by 93.10 % (83.10 %) on average, and the total number of bit flips of all PCM cells can be reduced by 64.00 % (70.90 %) on average. Furthermore, the results show that NV-FTL can evenly distribute write activities among PCM cells in comparison with a representative baseline FTL scheme.

## References

1. Duan R, Bi M, Gniady C (2011) Exploring memory energy optimizations in smartphones. In: Proceedings of the international green computing conference and workshops (IGCC '11), pp 1–8
2. Perrucci G, Fitzek F, Widmer J (2011) Survey on energy consumption entities on the smartphone platform. In: Proceedings of the IEEE 73rd vehicular technology conference (VTC '11), pp 1–6
3. Wong HSP, Raoux S, Kim S, Liang J, Reifenberg JP, Rajendran B, Asheghi M, Goodson KE (2010) Phase change memory,” Proc IEEE 98(12):2201–2227
4. Cho S, Lee H, 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 '09), pp 347–357

5. Hu J, Xue CJ, Zhuge Q, Tseng W-C, Sha EH-M (2013) Write activity reduction on non-volatile main memories for embedded chip multiprocessors. *ACM Trans Embed Comput Syst* 12(3):77:1–77:27
6. Lee BC, Ipek E, Mutlu O, Burger D (2009) Architecting phase change memory as a scalable DRAM alternative. In: *Proceedings of the 36th annual international symposium on computer architecture (ISCA '09)*, pp 2–13
7. Qureshi MK, Karidis J, Franceschini M, Srinivasan V, Lastras L, Abali B (2009) 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 '09)*, pp 14–23
8. Qureshi MK, Srinivasan V, Rivers JA (2009) Scalable high performance main memory system using phase-change memory technology. In: *Proceedings of the 36th annual international symposium on computer architecture (ISCA '09)*, pp 24–33
9. Dhiman G, Ayoub R, Rosing T (2009) PDRAM: a hybrid PRAM and DRAM main memory system. In: *Proceedings of the 46th annual design automation conference (DAC '09)*, pp 664–669
10. Ferreira AP, Zhou M, Bock S, Childers B, Melhem R, Mossé D (2010) Increasing PCM main memory lifetime. In: *Proceedings of the conference on design, automation and test in Europe (DATE '10)*, pp 914–919
11. Hosomi M, Yamagishi H, Yamamoto T, Bessho K, Higo Y, Yamane K, Yamada H, Shoji M, Hachino H, Fukumoto C, Nagao H, Kano H (2005) A novel nonvolatile memory with spin torque transfer magnetization switching: spin-RAM. In: *Proceedings of the IEEE international on electron devices meeting (IEDM '05)*, pp 459–462
12. Oboril F, Bishnoi R, Ebrahimi M, Tahoori M (2015) Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy. *IEEE Trans Comput Aided Des Integr Circuits Syst* 34(3):367–380
13. Wen W, Zhang Y, Chen Y, Wang Y, Xie Y (2014) PS3-RAM: a fast portable and scalable statistical STT-RAM reliability/energy analysis method. *IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD)* 33(11):1644–1656
14. Xue CJ, Zhang Y, Chen Y, Sun G, Yang JJ, Li H (2011) Emerging non-volatile memories: opportunities and challenges. In: *Proceedings of the seventh IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES+ISSS '11)*, pp 325–334
15. *International Technology Roadmap for Semiconductors (2007) Process integration, devices, and structures (2007 edition)*. <http://developer.intel.com>
16. Xie Y (2011) Modeling, architecture, and applications for emerging memory technologies. *IEEE Des Test Comput* 28(1):44–51
17. Chung T-S, Park D-J, Park S, Lee D-H, Lee S-W, Song H-J (2009) A survey of flash translation layer. *J Syst Archit* 55(5–6):332–343
18. Ban A (1995) Flash file system. US patent 5,404,485
19. Ban A (1999) Flash file system optimized for page-mode flash technologies. US patent 5,937,425
20. Wu C-H, Kuo T-W (2006) An adaptive two-level management for the flash translation layer in embedded systems. In: *Proceedings of the 2006 IEEE/ACM international conference on computer-aided design (ICCAD '06)*, pp 601–606
21. Chang Y-H, Hsieh J-W, Kuo T-W (2007) Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design. In: *Proceedings of the 44th annual conference on design automation (DAC '07)*, pp 212–217
22. Wang Y, Liu D, Wang M, Qin Z, Shao Z, Guan Y (2010) RNFTL: a reuse-aware NAND flash translation layer for flash memory. In: *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on languages, compilers, and tools for embedded systems (LCTES '10)*, pp 163–172
23. Wang Y, Liu D, Qin Z, Shao Z (2011) An endurance-enhanced flash translation layer via reuse for NAND flash memory storage systems. In: *Proceedings of the conference on design, automation and test in Europe (DATE '11)*, pp 1–6

24. Qin Z, Wang Y, Liu D, Shao Z (2011) A two-level caching mechanism for demand-based page-level address mapping in NAND flash memory storage systems. In: Proceedings of the 17th IEEE real-time and embedded technology and applications symposium (RTAS '11), pp 157–166
25. Qin Z, Wang Y, Liu D, Shao Z (2010) Demand-based block-level address mapping in large-scale NAND flash storage systems. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES/ISSS '10), pp 173–182
26. Qin Z, Wang Y, Liu D, Shao Z, Guan Y (2011) MNFTL: an efficient flash translation layer for MLC NAND flash memory storage systems. In: Proceedings of the 48th design automation conference (DAC '11), pp 17–22
27. Liu D, Wang Y, Qin Z, Shao Z, Guan Y (2011) A space reuse strategy for flash translation layers in SLC NAND flash memory storage systems. *IEEE Trans Very Large Scale Integr (VLSI) Syst* 20(6):1094–1107
28. Kim JK, Lee HG, Choi S, Bahng KI (2008) A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In: Proceedings of the 8th ACM international conference on embedded software (EMSOFT '08), pp 31–40
29. Wang J, Dong X, Xie Y, Jouppi N (2013) i2WAP: improving non-volatile cache lifetime by reducing inter- and intra-set write variations. In: IEEE 19th international symposium on high performance computer architecture (HPCA '13), pp 234–245
30. Joo Y, Niu D, Dong X, Sun G, Chang N, Xie Y (2010) Energy- and endurance-aware design of phase change memory caches. In: Proceedings of the conference on design, automation and test in Europe (DATE '10), pp 136–141
31. Qureshi M, Franceschini M, Lastras-Montano L (2010) Improving read performance of phase change memories via write cancellation and write pausing. In: IEEE 16th international symposium on high performance computer architecture (HPCA '10), pp 1–11
32. Hu J, Xue CJ, Tseng W-C, He Y, Qiu M, Sha EH-M (2010) Reducing write activities on non-volatile memories in embedded CMPs via data migration and recomputation. In: Proceedings of the 47th design automation conference (DAC '10), pp 350–355
33. Hu J, Xue CJ, Zhuge Q, Tseng W-C, Sha EH-M (2013) Write activity reduction on non-volatile main memories for embedded chip multiprocessors. *ACM Trans Embed Comput Syst* 12(3):77:1–77:27
34. Ferreira A, Childers B, Melhem R, Mosse D, Yousif M (2010) Using PCM in next-generation embedded space applications. In: 2010 16th IEEE real-time and embedded technology and applications symposium (RTAS '10), pp 153–162
35. Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E (2002) A survey on sensor networks. *IEEE Commun Mag* 40(8):102–114
36. Micron Technology, Inc. (2011) Micron phase change memory. <http://www.micron.com/products/pcm/>
37. Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. *Nature* 453:80–83
38. Zhou P, Zhao B, Yang J, Zhang Y (2009) A durable and energy efficient main memory using phase change memory technology. In: Proceedings of the 36th annual international symposium on computer architecture (ISCA '09), pp 14–23
39. Lee B, Zhou P, Yang J, Zhang Y, Zhao B, Ipek E, Mutlu O, Burger D (2010) Phase-change technology and the future of main memory. *IEEE Micro* 30(1):143–143
40. Yang B-D, Lee J-E, Kim J-S, Cho J, Lee S-Y, Yu BG (2007) A low power phase-change random access memory using a data-comparison write scheme. In: IEEE international symposium on circuits and systems (ISCAS '07), pp 3014–3017
41. DiskMon for Windows (2006) <http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx>