THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF COMPUTING

Write-Activity-Aware NAND Flash Memory Management for PCM-based Embedded Systems

By

DUO LIU

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

November 2011

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signature)

_____LIU Duo_____ (Name of Student)

ABSTRACT

Due to its properties of high density, in-place update, and low standby power, phase change memory (PCM) becomes a promising main memory alternative in embedded systems, and is recently introduced to embedded system designs. However, the endurance of PCM keeps drifting down and greatly limits the lifetime of the whole system. On the other hand, NAND flash memory is widely used as a secondary storage and has been integrated into PCM-based embedded systems. So this thesis targets at an embedded system with PCM and NAND flash memory. Since both NAND flash memory and PCM have limited lifetime, how to effectively manage NAND flash memory while considering PCM endurance is a challenge issue for PCM-based embedded systems.

To manage NAND flash memory, flash translation layer (FTL) is designed to emulate NAND flash memory as a disk drive, by mapping logical addresses to physical addresses in NAND flash memory at a granularity of page-level or block-level [37, 51]. Correspondingly, most of the proposed FTL techniques are mainly categorized into page-level or block-level based on the granularity of mapping unit [19]. As PCM-based main memory exhibits non-volatility feature, to obtain high access performance, FTL mapping table can be kept into PCM permanently without considering power failure. However, the frequently updated FTL mapping table imposes a large number of write activities in PCM, and may lead to a shortened PCM lifetime. Therefore, effective management techniques are needed to explore traditional page-level or block-level FTL designs and make them write activity aware, for enhancing the lifetime of the PCM-based embedded systems.

In this thesis, we focus on exploring the challenge issues imposed by the management of NAND flash memory in PCM-based embedded systems. Corresponds to the existing page-level and block-level FTL designs, we present for the first time three write-activity-

aware flash memory management techniques, to effectively manage NAND flash memory and enhance the lifetime of PCM-based embedded systems. To the best of our knowledge, this is the first work to study how to effectively manage NAND flash memory in PCM-based embedded systems by considering the endurance issue of PCM. We hope this work can serve as a first step towards the design of write-activity-aware flash memory management for PCM-based embedded systems.

Keywords: Phase change memory, PCM-based embedded system, NAND flash memory, flash translation layer, write activity, endurance.

PUBLICATIONS

Journal Papers

1. **Duo Liu**, Yi Wang, Zili Shao, Minyi Guo, Jingling Xue, "Optimally Maximizing Iteration-Level Loop Parallelism", IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 23, No. 3, pp. 564-572, March 2012.

2. **Duo Liu**, Yi Wang, Zhiwei Qin, Zili Shao, Yong Guan, "A Space Reuse Strategy for Flash Translation Layers in SLC NAND Flash Memory Storage Systems", Accepted in IEEE Transactions on Very Large Scale Integration Systems (TVLSI), 2011.

3. Yi Wang, **Duo Liu**, Zhiwei Qin, Zili Shao, "Optimally Removing Inter-Core Communication Overhead for Streaming Applications on MPSoCs", Accepted in IEEE Transactions on Computers (TC), 2011.

4. Miao Liu, **Duo Liu**, Yi Wang, Meng Wang, Zili Shao, "On Improving Real-Time Interrupt Latencies of Hybrid Operating Systems with Two-Level Hardware Interrupts", IEEE Transactions on Computers (TC), Volume 60, Number 7, pp. 978-991, July 2011.

5. Yi Wang, Hui Liu, **Duo Liu**, Zhiwei Qin, Zili Shao, E. H.-M. Sha, "Overhead-Aware Energy Optimization for Real-Time Streaming Applications on Multiprocessor System-on-Chip", ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 16, Issue 2, pp. 14:1-14:32, March 2011.

6. Meng Wang, Yi Wang, **Duo Liu**, Zhiwei Qin, Zili Shao, "Compiler-Assisted Leakage-Aware Loop Scheduling for Embedded VLIW DSP Processors", Elsevier Journal of Systems and Software (JSS), Volume 83, Issue 5, pp. 772-785, May 2010.

Conference Papers

1. **Duo Liu**, Tianzheng Wang, Yi Wang, Zhiwei Qin, Zili Shao, “A Block-Level Flash Memory Management Scheme for Reducing Write Activities in PCM-based Embedded Systems”, in the 15th Design, Automation and Test in Europe (DATE 2012), March 2012, Dresden, Germany.

2. **Duo Liu**, Tianzheng Wang, Yi Wang, Zhiwei Qin, Zili Shao, “PCM-FTL: A Write-Activity-Aware NAND Flash Memory Management Scheme for PCM-based Embedded Systems”, in the 32nd IEEE Real-Time Systems Symposium (RTSS 2011), Vienna, Austria, Nov. 29-Dec. 2, 2011.

3. **Duo Liu**, Zili Shao, Meng Wang, Minyi Guo, Jingling Xue, ”Optimal Loop Parallelization for Maximizing Iteration-Level Parallelism”, in International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2009), Grenoble, France, Oct. 2009.

4. Zhiwei Qin, Yi Wang, **Duo Liu** and Zili Shao, ”Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems”, 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS2012) in conjunction with Cyber-Physical Systems Week (CPSWEEK 2012), Beijing, China, April 16-19, 2012.

5. Tianzheng Wang, **Duo Liu**, Zili Shao, Chengmo Yang, “Write-Activity-Aware Page Table Management for PCM-based Embedded Systems”, in the 17th Asia South Pacific Design Automation Conference (ASP-DAC 2012), Sydney, Australia , Jan. 30-Feb. 2, 2012.

6. Yi Wang, **Duo Liu**, Zhiwei Qin, Zili Shao, “An Endurance-Enhanced Flash Translation Layer via Reuse for NAND Flash Memory Storage Systems”, in the 14th Design, Automation and Test in Europe (DATE 2011), pp. 14-20, March 2011, Grenoble, France.

7. Yi Wang, **Duo Liu**, Zhiwei Qin, Zili Shao, “Memory-Aware Optimal Scheduling with Communication Overhead Minimization for Streaming Applications on Chip Multiprocessors”, in the 31st IEEE Real-Time Systems Symposium (RTSS 2010), pp. 350-359, November 2010, San Diego, CA, USA.

8. Yi Wang, **Duo Liu**, Meng Wang, Zhiwei Qin, Zili Shao, “Optimal Task Scheduling by Removing Inter-core Communication Overhead for Streaming Applications on MPSoC”, in the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010), pp. 195-204, April 2010, Stockholm, Sweden.

9. Yi Wang, **Duo Liu**, Meng Wang, Zhiwei Qin, Zili Shao, Yong Guan, “RNFTL: A Reuse-Aware NAND Flash Translation Layer for Flash Memory”, in ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2010), pp. 163-172, April 2010, Stockholm, Sweden.

10. Zhiwei Qin, Yi Wang, **Duo Liu**, Zili Shao, Yong Guan, “MNFTL: An Efficient Flash Translation Layer for MLC NAND Flash Memory Storage Systems”, in the 48th IEEE/ACM Design Automation Conference (DAC 2011), pp. 12-18, June 2011, San Diego, CA, USA.

11. Zhiwei Qin, Yi Wang, **Duo Liu**, Zili Shao, “A Two-Level Caching Mechanism for Demand-Based Page-Level Address Mapping in NAND Flash Memory Storage Systems”, in the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011), pp. 157-166, April 2011, Chicago, IL, USA.

12. Zhiwei Qin, Yi Wang, **Duo Liu**, Zili Shao, “Demand-Based Block-Level Address Mapping in Large-Scale NAND Flash Storage Systems”, in the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES +ISSS 2010), pp. 173-182, October 2010, Scottsdale, Arizona, USA.

13. Meng Wang, Yi Wang, **Duo Liu**, Zili Shao, “Improving the Reliability of Embedded Systems with Cache and SPM”, in the 2009 IEEE International Symposium on Trust,

Security and Privacy for Pervasive Applications in conjunction with the 2009 IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2009), pp. 825-830, October 2009, Macau.

14. Meng Wang, **Duo Liu**, Yi Wang, Zili Shao, “Loop Scheduling with Memory Access Reduction under Register Constraints for DSP Applications”, in the 2009 IEEE Workshop on Signal Processing Systems (SiPS 2009), pp. 139-144, October 2009, Tampere, Finland.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Zili Shao, who offered me the opportunity to pursue my PhD study with a group of talented and energetic people. I am really impressed by his vast knowledge and skill in many areas and his professional supervision. His expertise, thoughtfulness, and endless patience, added considerably to my research experience. It is my great pleasure to be a student of Dr. Shao in my life, and I would like to thank him for providing me a platform to conduct independent research, and training me to be a good researcher over years. I greatly appreciate his advice on research, writing, teaching and presentation. Without his encouragement and help during the difficult times in my PhD study, this body of work would not have been possible.

I would like to thank the other members of Dr. Shao's research group - Yi Wang, Zhiwei Qin, Tianzheng Wang, Meng Wang, Guohui Wang, and Chunjing Mao - not only for their kindly help and corporation during these years, but also for the friendly and relaxing working environment created by them. I will never forget the days and nights we work and discuss together. A special thank you to Yi Wang who was always willing and ready to help me at any time. I am also deeply indebted to Tianzheng Wang who spent countless hours for assisting me on the collection of experimental results. I appreciate Zhiwei Qin, Guohui Wang and Chunjing Mao for their constructive suggestions on research ideas during my PhD study. I wish to them all the success.

I also would like to thank all my teachers from whom I learned so much in my long journey of formal education. Specially thanks go to Prof. Jiannong Cao, Dr. Zhijun Wang, Dr. Bin Xiao, Dr. Yan Liu, Dr. Lei Zhang, and Dr. Alvin Chan at the Hong Kong Polytechnic University.

I would like to thank all the visitors who share their research experiences and com-

ment our research work when they visited our research group. I have had a deeper understanding of research thanks to their suggestions. Especially thanks to Prof. Tei-wei Kuo, Prof. Jingling Xue, Prof. Nikil Dutt, Dr. Yiran Chen, and Prof. Lui Sha. They broaden my scientific outlook by providing me with multiple new ideas and research philosophy. I will not forget the seminar presented by Prof. Kuo, who first time introduced the research of flash memory management to our group. I also deeply thank to Prof. Xue for his help and guidance on the research of loop parallelization. I would like to acknowledge Prof. Dutt for his constructive comments on my research work. Thanks go to Dr. Chen for his introduction of the emerging memory technologies in our group, especially the phase change memory. I appreciate Prof. Sha for his extent of knowledge and experience by sharing the research philosophy with us.

Also, I wish to acknowledge my appreciation to Yufei Wang, SZETO Chi Cheong, Haomian Zheng, Guobin Liu, Jin Xie, Qingjun Xiao, Jiaqing Luo, Xiaopeng Fan, Dongmin Guo, Xiaocui Sun, Jinjiang Yu and Baozhuang Niu who exhibited their friendship and shared with me the pleasure of the Ph.D. study at the Hong Kong Polytechnic University.

Special thanks to my wife, Liang Liang, who witnesses the joys and sorrows of my PhD study miles away. Though we are separated by mountains and rivers, I am grateful for her endless love, patience, understanding and support, without which I could not have an enjoyable research life.

Finally, but most significantly, I deeply thank my family for their long-term caring and encouragement through my entire life, for letting me pursue my dream for so long and so far away from home, and for giving me the motivation to finish this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

In the last decade, various emerging non-volatile memory technologies, such as phase change memory (PCM), spin torque transfer RAM (STT-RAM), magnetic RAM (MRAM), and ferroelectric RAM (FRAM), have been developed and considered as a replacement for DRAM. Among them, PCM is known to be one of the most promising technologies, due to its high density, in-place update, and low standby power. Therefore, PCM is considered as a lead alternative of DRAM (dynamic random access memory), and has been used as a main memory with a small-sized DRAM cache in embedded systems [21,26,77,96]. However, compared to DRAM, PCM can only sustain limited write operations (10^6 to 10^8 bit flips per cell) [38]. As main memory is a frequently accessed component, it is necessary to reduce redundant write activities in PCM to enhance the reliability of PCM-based embedded systems. On the other hand, with the advantages of small size, shock resistance, and low power, NAND flash memory is widely used as a secondary storage and has been integrated into PCM-based embedded systems [48, 68, 87]. How to avoid a fast worn-out of such emerging embedded systems and effectively manage NAND flash memory should be taken into account. Therefore, this thesis 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.

Several techniques have been recently proposed to enhance the lifetime of PCM at architectural/hardware level. In [52, 101], Zhou et al. propose a redundant bit removal technique, by which a write to PCM is ignored if its designated PCM cell holds the same value. In [77], Qureshi et al. propose Start-Gap to evenly distribute write activities among all PCM cells for enhancing the lifetime of PCM-based main memory. In [21], Dhiman et al. introduce a scheme wherein a page manager is developed to allocate pages across PCM and DRAM for improving PCM lifetime. In [87], Sun et al. propose a hybrid storage archi-

tecture, wherein PCM lifetime is prolonged by inserting released log sectors to the list of free sectors based on its number of writes recorded. In [43], Joo et al. propose an energy- and endurance-aware PCM cache design which reduces write activities by read-before-write and data inverting techniques. In [88], Sun et al. reduce write intensity in PCM by storing frequently written values in compressed form.

On the other hand, some software level techniques have also been developed, such as the code optimization techniques [33] and write-aware scheduling techniques [34] proposed by Hu et al. Similarly, Ferreira et al. [26] propose three schemes, i.e., write minimization, unnecessary writes reduction, and a wear-leveling scheme, to increase the lifetime of PCM-based main memory. However, most of the hardware/software techniques do not consider the write activities caused by the management procedure of other devices, such as NAND flash memory. As NAND flash memory has already been used in PCM-based embedded systems [48,68,87], some redundant write activities during the management process of NAND flash memory can lead to a lifetime degradation of PCM-based main memory. Therefore, unlike previous work, the work proposed in this thesis can make the NAND flash memory management scheme write activity aware, for enhancing the lifetime of PCM-based embedded systems.

In PCM-based embedded systems, to use NAND flash memory, flash translation layer (FTL) is designed to emulate NAND flash memory as a disk drive, and logical addresses are mapped to physical addresses in NAND flash memory at a granularity of page-level or block-level [37, 51]. Following I/O requests, an FTL mapping table is employed to keep track of the continually updated mapping records. Many FTL schemes have been proposed [5, 6, 18, 29, 93], and most of them are mainly categorized into page-level scheme or block-level scheme according to the granularity of mapping unit [19]. To provide fast lookup and high data access performance, 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 traditional 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. [48] propose a page-level FTL, namely *h*FTL, in which page-level FTL mapping table is kept in PCM and user data is stored in NAND flash memory. Nevertheless, *h*FTL does not consider redundant write activities occurred in PCM because of the frequently updated FTL mapping table, which may lead to a shortened PCM lifetime. As the lifetime of PCM is mainly determined by the maximum number of bit flips in each PCM cell, it is important to reduce the maximum number of bit flips in each PCM cell to enhance the reliability of the entire system. New techniques, therefore, are needed to explore traditional page-level or block-level FTL designs and make them write activity aware, for reducing unnecessary write activities and enhancing the lifetime of the PCM-based embedded systems.

In this thesis, we focus on exploring the challenge issues imposed by the management of NAND flash memory in PCM-based embedded systems. Corresponds to the existing page-level and block-level FTL designs, we propose three write-activity-aware NAND flash memory management techniques for reducing 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, with the advantage that no changes are required to the file system, and hardware implementation of the NAND/PCM chip. Note that the mapping records inside FTL mapping table are represented in a binary form in PCM. Therefore, the objective 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 the best of our knowledge, this is the first work to study how to effectively manage NAND flash memory in PCM-based embedded systems by considering the endurance issue of PCM. We hope this work can serve as a first step towards the design of write-activity-aware flash memory management for PCM-based embedded systems.

The rest of this chapter is organized as follows: Section 1.1 presents the related work. Section 1.2 presents the unified research framework. Section 1.3 summarizes the contributions of this thesis. Section 1.4 gives the outlines of the thesis.

1.1 Related Work

In this section, we outline previous approaches related to PCM-based embedded systems, write activity reduction for PCM, and some related FTL schemes. We briefly describe these approaches, and detailed comparisons with representative techniques are presented in respective chapters.

1.1.1 PCM-based Embedded Systems

As an emerging non-volatile memory, PCM is considered as a promising candidate of traditional memories at various levels in current memory hierarchy. Compared to DRAM, the read/write latency and power consumption of PCM are slightly worse than those of DRAM [38]. However, PCM provides a significant density over DRAM, which means more memory capacity and lower price per memory cell within the same chip area. Besides, PCM offers negligible idle power than that of DRAM. On the other hand, unlike NAND flash memory, PCM supports in-place-update regardless of the erase-before-write constraint of NAND flash memory. The read/write latency of PCM is much better than that of NAND flash memory. In addition, the lifetime of PCM is three orders of magnitude longer than that of NAND flash memory.

Therefore, because of its attractive advantages, PCM is being incorporated into embedded systems [24], e.g., Samsung GT-E2550 GSM mobile handsets [50, 81, 86], and is considered as a replacement of DRAM to achieve larger main memory capacity [20, 53, 78]. Even though PCM is sightly slower than DRAM and is constrained by its limited lifetime, with clever optimizations at software/hardware level, such as buffering frequently accessed data by a small-sized DRAM cache, it is feasible to use PCM-based main memory in embedded systems [21, 26, 69, 77, 78, 96].

Figure 1.1 shows a typical PCM-based embedded system, which consists of a hybrid PCM-based main memory and a NAND flash memory proposed by [48]. To obtain a best capacity and latency, the hybrid main memory adopts a large-sized PCM and a small-sized

DRAM cache. 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 to improve performance and PCM lifetime. In the system, NAND flash memory is employed as a secondary storage for storing user data that are accessed by file systems.

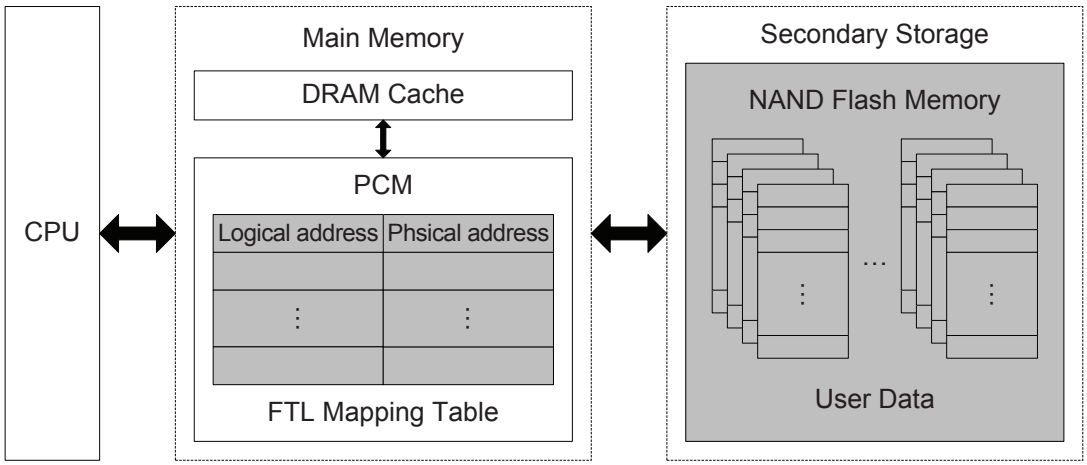


Figure 1.1. PCM-based Embedded Systems.

To conceal the unfavorable characteristics of NAND flash memory, an intermediate software module called flash translation layer (FTL) is employed to emulate NAND flash memory as a block device [37]. The main role of FTL is to redirect logical addresses from the file systems of a host into physical addresses in NAND flash memory, and maintains a mapping table to keep track of the mapping information. FTL not only supports address translation but also provides other useful components such as garbage collector and wear-leveler that are used to optimize the space utilization and maintain the same level of wear for each block in NAND flash memory. Following I/O requests to NAND flash memory, 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 FTL scheme write activity aware in PCM-based embedded systems.

Figure 1.2 shows a software-level architecture of the incorporated flash translation layer module [51]. In this architecture, flash translation layer provides three components: address translator [6], garbage collector [10], and wear-leveler [12]. In FTL, address translator maintains an FTL mapping table, which usually located in main memory could translate addresses between logical address and physical address; garbage collector reclaims space by erasing obsolete blocks in which there exists invalid data; wear-leveler is an optional component that distributes erase operations evenly across all blocks, so as to extend the lifetime of NAND flash memory. This thesis focuses on improving the management of address translator in flash translation layer, to reduce write activities in the PCM-based embedded systems.

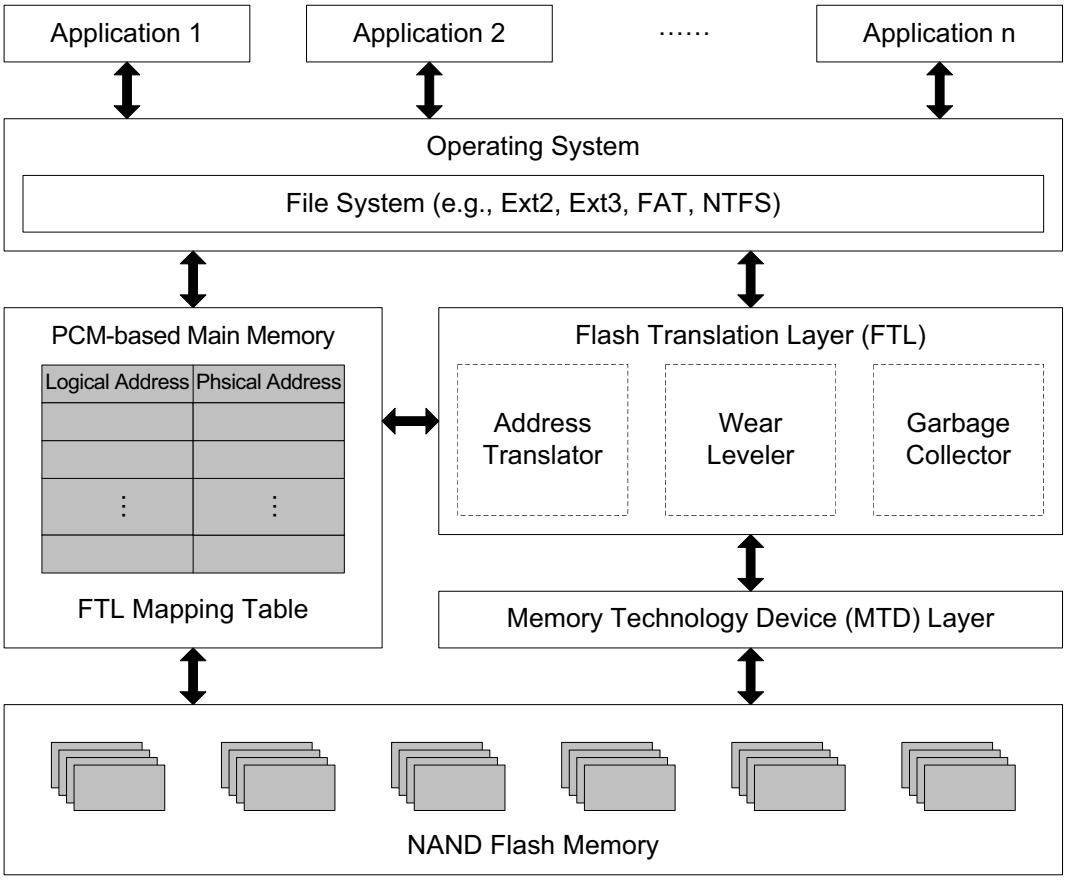


Figure 1.2. A typical management architecture of NAND flash memory with flash translation layer.

1.1.2 Write Activity Reduction for PCM

To incorporate PCM into main memory hierarchy, one most important challenge is that PCM can only suffer limited write cycles, and thus may wear out earlier than DRAM-based main memory. Therefore, to mitigate this limitation, extensive work recently has been done to reduce write activities for PCM-based embedded systems. The research of reducing write activities in PCM can be mainly classified into two categories: architectural/hardware level and software level.

The Architectural/Hardware Level

At the architectural/hardware level, to extend PCM lifetime, various techniques have been proposed to reduce PCM write activities and perform wear leveling.

To reduce write activities, several techniques have been proposed including differential write [101], Compression [88, 99], Flip-N-Write [16], and row-buffer locality-aware data placement [98]. In [52, 101], Zhou et al. propose a suit of hierarchical techniques: redundant bit-write removal, row shifting, and segment swapping, to improve the lifetime of PCM-based main memory. Write activity reduction is accomplished mainly based on the idea of data-comparison write (DCW), by which a write to PCM is ignored if its designated PCM cell holds the same value. For example, only the right most bit '0' is written into PCM if a value of '1010' to be written into a PCM destination with '1011'. In [77, 78], Qureshi et al. introduce a PCM-based hybrid memory architecture wherein PCM is employed as a main memory while a small-sized DRAM is employed as a cache buffer. Based on this architecture, Start-Gap is proposed to evenly distribute write operations across all PCM cells, and a line-level write scheme is developed to write only the dirty lines in the cache buffer into the PCM, for improving wear leveling and enhancing the lifetime of PCM-based main memory. In [21], Dhiman et al. explore the challenges after incorporating PCM into the hybrid main memory hierarchy with DRAM. To improve the PCM lifetime, a book keeping hardware technique is proposed to store write frequency information into PCM at a page level granularity. In [16], by extending idea of data-comparison write technique pro-

posed by [101], Cho et al. propose a simple architectural technique to replace a PCM write operation with a more efficient read-modify-write operation for reducing redundant bit programming. In [87], Sun et al. propose a hybrid solid storage architecture that uses PCM as a log area, wherein PCM lifetime is prolonged by inserting released log sectors to the list of free sectors based on its number of writes recorded. In [43], Joo et al. propose an energy- and endurance-aware PCM cache design which reduces write activities by read-before-write and data inverting techniques. In [88], Sun et al. propose a frequent-value based data storage architecture, wherein write intensity in PCM is reduced by storing frequently written values in compressed (encoded) form.

The wear leveling techniques that evenly distribute write activities have been proposed including hot/cold line shifting and segment swapping [101], randomized mapping such as Start-Gap [77] and Security Refresh [83], and adaptive wear leveling with online attack detector [75]. To defend against malicious attacks, the randomized approaches have been proposed to randomly distribute writes [75, 77, 83, 85], and the wear-leveling-aware encryption techniques have been proposed in [14, 49].

To deal with process variation, a scheme with process-variation-aware current provision, adaptive page-level data comparison writes, and dirty-cache-line compression is proposed in [99], and a fine-grained current provision and voltage upscalling scheme is proposed in [39]. To solve the problems caused by resistance drift, several schemes have been proposed including adaptive data inversion/rotation scheme based on different resistance-drift-sensitive patterns and resistance-drift-aware SLC/MLC reconfiguration [100], and drift-tolerant coding based on modulation coding by utilizing relative order of resistance levels in a codeword [66].

The Software Level

On the other hand, to enhance PCM lifetime, some techniques have also been developed at the software level. In [33], an embedded chip multiprocessors (CMPs) system with scratch pad memory (SPM) and PCM-based main memory is explored. In this system, a data mi-

gration and code optimization techniques are proposed to avoid write-backs of shared data, for extending the lifetime of PCM-based main memory. In [34], Hu et al. propose two optimization techniques, write-aware scheduling and re-computation, to schedule the tasks in the program with the consideration of write activities in main memory, for minimizing write activities in non-volatile memories such as PCM. Similarly, in [26], Ferreira et al. develop three schemes, i.e., write minimization, unnecessary writes reduction, and a wear leveling scheme, to improve the lifetime of PCM-based main memory. Besides, in [27], they propose a page partitioning technique and a clean-preferred page replacement algorithm to reduce the number of write-back data to PCM, for enhancing PCM lifetime. In [21], based on the hybrid main memory and the hardware book keeping technique, Dhiman et al. introduce an efficient OS-level page manager to evenly allocate pages across PCM and DRAM for improving PCM wear leveling. In [22], Dong et al. study the endurance variation of PCM cells, and propose a variant of wear leveling mechanism, through physical address re-mapping and data swapping, to balance wear rates of PCM cells across the whole PCM chip. In [8], Bock et al. introduce the concept of useless write-back data that is not used again by the program, and develop an analytical framework to determine the number of useless write-backs, to improve the lifetime of the PCM-based main memory.

However, most of the previous hardware/software techniques do not consider the write activities caused by the management procedure of other devices in the PCM-based embedded systems, such as NAND flash memory. As NAND flash memory has already been used in PCM-based embedded systems as a secondary storage [48, 68, 87], some redundant write activities during the management process of NAND flash memory can lead to a lifetime degradation of PCM-based main memory. Therefore, unlike previous work, the work proposed in this thesis can make the NAND flash memory management write activity aware, for enhancing the lifetime of PCM-based embedded systems.

1.1.3 FTL Schemes

NAND flash memory has been widely used in various applications. A lot of designs and implementations of NAND flash memory management have been proposed in the literature. As FTL plays a critical role in NAND flash memory management, different FTL schemes have been proposed and can be categorized into three major types: page-level mapping, block-level mapping, and hybrid-level mapping. Hybrid-level FTL overcomes the shortcomings of page-level mapping and block-level mapping, and provides a balance between space overheads and flexibility. Therefore, hybrid-level FTL has been widely adopted in NAND flash memory designs and implementations, especially for large-scale flash storage systems [11]. In particular, Wu and Kuo [93] describe an adaptive hybrid-level approach that can dynamically and adaptively switch between page-level and block-level in the mapping of logical block addresses into physical block addresses. Choudhuri and Givargis [17] employ a lookup table and page cache method into the translation layer to speedup address translation and improve read and write throughput. Wu et al. [94] use a search-tree-like caching mechanism and a replacement strategy for efficient address translation. Park et al. [67] apply a flash translation layer architectural framework to decide which configuration of address mapping parameters yields the best performance.

In recent studies, several schemes have been proposed for log-block-based hybrid-level FTL schemes. In log-block-based FTL schemes, a limited number of log blocks are provided for all data blocks to store updated data. Kim et al. [47] proposed a hybrid-level scheme, called log-block-based FTL (BAST), in which all data blocks share a set of log blocks for update operations. Based on BAST, several log-block-based FTL schemes have been proposed to explore the block associativity of each log block. Lee et al. proposed a fully-associative mapping scheme called FAST [58]. More recently, Cho et al. propose a K-Associate log-block-based scheme called KAST [15], which can limit the maximum log block associativity.

To balance the trade-off between page-level FTL and block-level FTL, some recent studies consider configurable or demand-based mapping scheme to reduce the size of map-

ping table and provide the flexibility of mapping scheme. Gupta et al. propose a demand-based caching scheme, called DFTL [29], to reduce page-level address mapping table. Yongsoo Joo et al. [41, 42] propose an online demand paging scheme that can fully exploit the eXecution-in-Place (XIP) capability of OneNAND flash. More recently, Qin et al. [70] propose a demand-based block-level address mapping scheme in large-scale NAND flash storage systems. Chang and Kuo [13] propose a commitment-based management strategy to improve the reliability of NAND flash memory. A three-level address translation architecture with an adaptive block mapping scheme is proposed. The proposed technique can accelerate the address translation process with the considerations of the limited RAM space. Hsieh et al. [32] propose a configurable mapping scheme that can trade-off the main-memory overhead and the system performance. In their scheme, the mapping between the virtual address to the physical address provides the flexibility to prevent a block from being used by any fixed physical block.

Some studies consider the system requirements and provide solutions for application specific flash memory management designs. Chu et al. [18] propose a set-based mapping strategy that can utilize thrown-away flash memory chips into downgraded products. Yongsoo Joo et al. [40,42] discovered programming energy variation of MLC NOR flash memory, and proposed an energy-aware data compression method to minimize the flash programming energy. Wu et al. [95] propose a file-system-aware flash translation layer, in which a filter mechanism is adopted to analyze the access requests and separate the metadata of file system and the ordinary files. Huang et al. [35] analyze the behavior and access pattern of flash memory storage system. Li et al. [59] propose a StableBuffer solution for flash devices that exploit write patterns of flash devices to optimize the performance of DBMS applications. Lee and Alex [56, 57] propose highly effective application specific embedded systems using NAND flash as primary memory and low latency instruction cache. On et al. [63] study the buffer management for flash-based databases, with a focus on addressing the read-write asymmetry and workload dynamics, and propose FD-Buffer that automatically adapts to the flash disk characteristics and the runtime workload. Hsieh et al. [31] propose a new architecture that utilizes the flash memory as a cache layer for disks to save energy consumption.

The above work presents excellent designs for different application specific architectures.

Most of the previous work provides good solution and improves the performance of FTL. However, no work targets at the emerging PCM-based embedded systems, wherein the NAND flash memory is used as a secondary storage. Though Kim et al. [48] propose a page-level FTL, namely *h*FTL, and focuses on the PCM-based embedded systems, in which page-level FTL mapping table is kept in PCM-based main memory and user data is stored in NAND flash memory. But *h*FTL does not consider redundant write activities occurred in PCM because of the frequently updated FTL mapping table, which may lead to a shortened PCM lifetime. Considering the lifetime limitation of PCM and the frequently accessed FTL mapping table in the PCM-based main memory, it is important to study the write activity of FTL schemes for the emerging PCM-based embedded systems. Therefore, different from the previous work, this thesis targets at the PCM-based embedded systems, and takes a first step to propose techniques for making the basic FTL design (page-level and block-level scheme) write activity aware, such that the lifetime of PCM-based embedded systems can be enhanced.

1.2 The Unified Research Framework

In this section, we present the unified research framework for the proposed techniques. Figure 1.3 illustrates the sketch of our research framework.

In this thesis, we target at the PCM-based embedded systems, wherein the PCM-based main memory with a small-sized DRAM cache is used for maintaining frequently accessed OS pages and the FTL mapping table, and a NAND flash memory is adopted as a secondary storage for storing user data accessed by file systems.

In this thesis, three flash memory management techniques considering write activity reduction for PCM are presented to improve the lifetime and performance of the PCM-based embedded systems, in terms of the granularity of FTL mapping unit. As shown in Figure 1.3, the traditional FTL design is redesigned by the proposed techniques in PCM-based

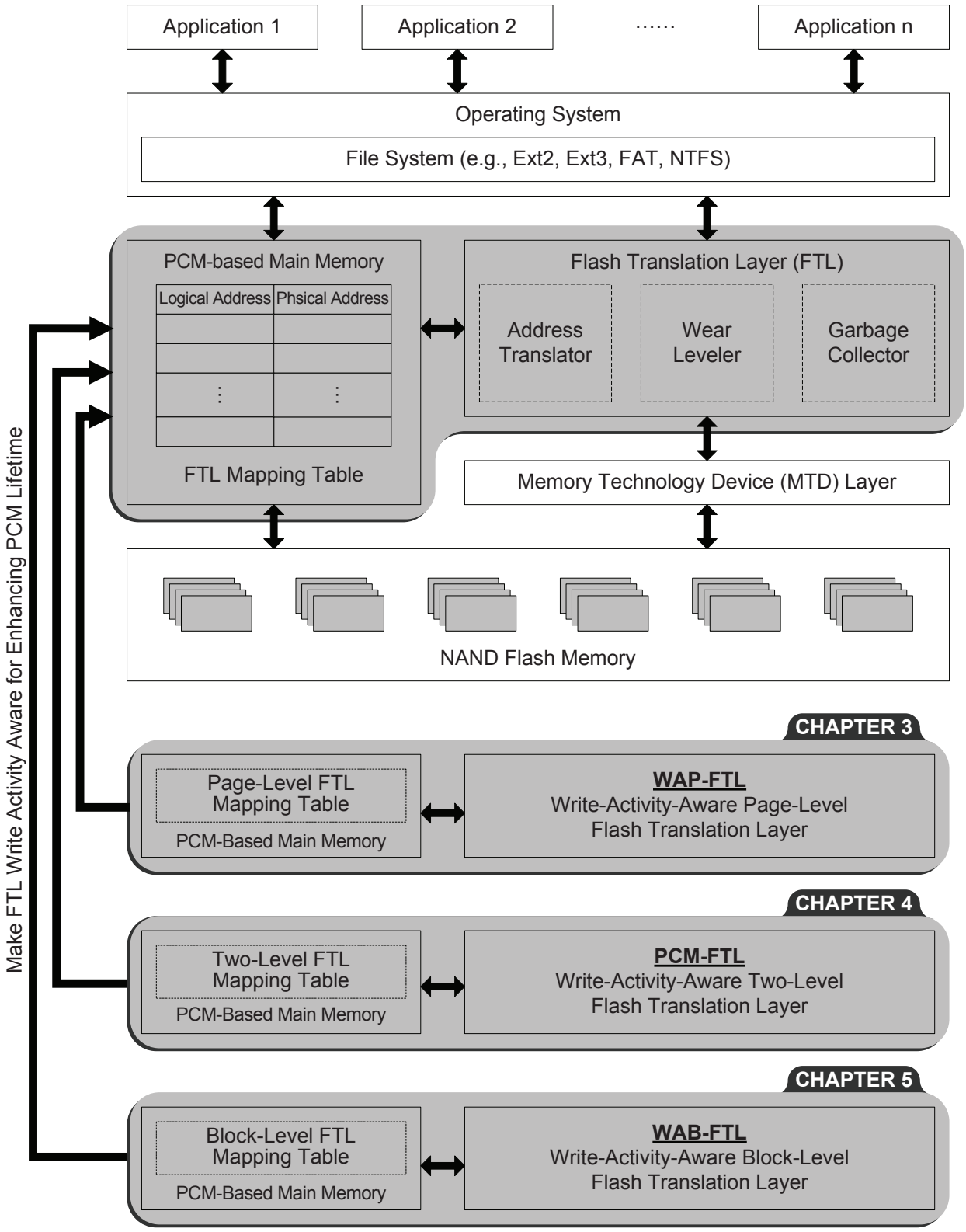


Figure 1.3. The Unified Research Framework.

embedded systems. Each of the proposed write-activity-aware FTL techniques maintains a corresponding FTL mapping table in PCM-based main memory. To effectively manage NAND flash memory while reducing write activities for the corresponding FTL mapping table in PCM, three write-activity-aware flash memory management techniques are proposed in this thesis.

- For the first technique, in Chapter 3, we first propose a **W**rite-**A**ctivity-aware **P**age-level **FTL** mapping technique, named WAP-FTL. In the WAP-FTL, we consider to reduce write activities for a page-level FTL mapping table in PCM. To achieve this, a write-activity-aware strategy is employed to prevent each bit in page-level FTL mapping table that hosted by PCM from being inverted frequently. Once a write request arrives in NAND flash memory, unlike the traditional page-level FTL design [5,48], the proposed page-level FTL technique, WAP-FTL, can actively choose a physical page whose physical address effects the minimum number of bit flips in FTL page-level mapping table, so as to effectively reduce write activities on PCM cells. However, the proposed WAP-FTL does not consider the access behavior of I/O requests, and the evaluation results also show that write activity in PCM will increase due to extra overhead of page copy introduced by the garbage collection.

- For the second technique, in Chapter 4, by extension the idea of WAP-FTL, we further propose a two-level FTL mapping technique, named PCM-FTL, which not only focuses on minimizing the write activities of PCM but also considering the access behavior of I/O requests. To achieve this, in PCM-based main memory, we propose a page-level FTL mapping table to handle not frequently updated random requests, and allocate a tiny mapping buffer of block-level FTL mapping table to record most frequently updated sequential requests. Similar to that of WAP-FTL, to further minimize write activities in PCM, PCM-FTL actively chooses a physical block in NAND flash memory whose physical block number incurs minimum number of bit flips.

- For the third technique, in Chapter 5, we present a **W**rite-**A**ctivity-aware **B**lock-level **FTL** mapping technique, named WAB-FTL, to effectively manage NAND flash mem-

ory while reducing write activities of the PCM-based embedded systems. Unlike WAP-FTL and PCM-FTL which require significant capacity in PCM for storing the larger page-level FTL mapping table, WAB-FTL is motivated that block-level FTL with much less memory requirement is more applicable for PCM-based embedded systems, as the capacity of current PCM prototype chips is very small and may not be practical for storing large table [61]. Therefore, in WAB-FTL, a block-level FTL mapping mechanism with write activity consideration is proposed. To reduce redundant write activities for PCM, 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.

In this thesis, we evaluate the proposed techniques WAP-FTL, PCM-FTL and WAB-FTL using a variety of realistic I/O traces, which reflect the realistic workloads of the system in accessing the secondary storage for daily use. Several real-life traces are obtained from DiskMon [1] running on the notebook with an Intel Pentium Dual Core 2GHz processor, a 200GB hard disk, and a 2GB DRAM. Besides, some other traces are collected from Google Android™2.3 [28] with Android Emulator (included in Android SDK). We modified the Linux kernel shipped with Android to record I/O requests in system log. Traces are gathered by *Android Debug Bridge* in Android SDK from the emulator to host computer. In order to reveal the actual impacts of the experimental schemes, we collected traces under heavy-loaded environment by using *Monkey*, which is an automatic stress test tool provided by Android SDK. With applications specified, it generates random events for them and send the events to the emulator for execution. The evaluation is conducted by a trace-driven simulation. We have developed a simulator, which simulates a PCM-based embedded systems with 1GB/4GB NAND flash memory, to evaluate our write-activity-aware flash memory management techniques against with the representative baseline FTL designs.

1.3 Contributions

The contributions of this thesis are summarized as follows.

- The major contribution of this thesis is the idea of considering write activity of NAND flash memory management schemes in PCM-based embedded systems. To reduce write activities of FTL mapping table and extend PCM lifetime, this thesis presents for the first time three write-activity-aware flash memory management techniques. For these techniques, the idea of reducing write activities is to actively find physical pages or blocks whose page number or block number incurs the minimum number of bit flips in PCM during the update process of page- or block-level FTL mapping table. Moreover, WAP-FTL demonstrates that write activities of PCM cannot be reduced by merely actively finding physical pages without considering the behavior of I/O requests. Based on this observation, by carefully studying the I/O request behavior, the other two techniques PCM-FTL and WAB-FTL provide good performance for reducing write activities in the PCM-based embedded systems.

- We propose a write-activity-aware page-level flash memory management technique, WAP-FTL to reduce write activities on a page-level FTL mapping table in PCM. WAP-FTL is mainly based on the idea of actively choosing a physical page whose physical page number causes the minimum number of bit flips in PCM. Though the evaluation results of WAP-FTL finally show that write activities in PCM will increase during garbage collection, WAP-FTL provides the potential research direction and suggests that the behavior of I/O requests must be taken into account.

- We present for the first time a write-activity-aware two-level flash management technique, PCM-FTL, which not only extends the work of WAP-FTL but also considers the access behavior of I/O requests. In PCM-FTL, a tiny mapping buffer of block-level FTL mapping table is allocated to record most frequently updated sequential requests, while a page-level FTL mapping table is used to handle not frequently updated random requests. Compared with a representative FTL scheme, the experimental results show that PCM-FTL can achieve an average reduction of 93.10% and a maximum reduction of 98.98% in the maximum number of bit flips for a PCM-based embedded system with 1GB NAND flash memory. In addition, the results also show that PCM-FTL

can achieve an even distribution of bit flips in PCM in comparison with the baseline scheme.

- We present for the first time a write-activity-aware block-level flash memory management technique, WAB-FTL, to effectively manage NAND flash memory while reducing write activities of the PCM-based embedded systems. With the advantage of much less memory requirement, in WAB-FTL, a new merge strategy (Lazy-Merge) and an additional tiny buffer (Cooling-Pool) are proposed, to make our approach write activity aware, such that the PCM lifetime is enhanced. Compared with the previous work, experimental results show that our technique could effectively reduce write activities in PCM-based embedded systems. Moreover, the experimental results also show that WAB-FTL can achieve an even distribution of bit flips in PCM cells in comparison with the baseline scheme.

- A trace-driven simulation framework is implemented, to evaluate the proposed write-activity-aware flash memory management schemes in the PCM-based embedded systems. We conducted experiments and compared with representative FTL schemes. Experimental results prove the effectiveness of the proposed schemes using a set of realistic I/O workloads.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

- In Chapter 2, we briefly introduce the background knowledge of phase change memory, NAND flash memory and flash translation layer.

- In Chapter 3, we present our write-activity-aware page-level flash memory management technique, WAP-FTL, and demonstrate its limitation in write activity reduction for PCM-based embedded systems.

- In Chapter 4, we present our write-activity-aware two-level flash memory management technique, PCM-FTL, to extend the work of WAP-FTL. We also show that PCM-FTL can effectively reduce write activities for PCM-based embedded systems.
- In Chapter 5, we present our write-activity-aware block-level flash memory management technique, WAB-FTL, and show that WAB-FTL with much less memory requirement can also effectively reduce write activities for PCM-based embedded systems.
- In Chapter 6, we present conclusions and discuss possible future directions of research arising from this work.

CHAPTER 2

BACKGROUND

In this chapter, we first present the background knowledge of phase change memory. Then we briefly introduce the characteristics of NAND flash memory. Finally, we discuss different types of flash translation layer that used for managing NAND flash memory in embedded systems.

2.1 Phase Change Memory

Phase change memory (PCM), also known as PCRAM, PRAM or Chalcogenide RAM, is a type of no-volatile memory techniques. PCM was first demonstrated in 1960s, it stores data by programming the resistance of chalcography alloy [64, 65, 96]. Due to material quality, power consumption and manufacture cost issues, the development of PCM technology is slow during the past decades. However, after adopting the new phase change material, such as $Ge_2Sb_2Te_5$ (GST) [97], PCM technology exhibits attractive advantages of bit-addressability, superior scalability, high density, low standby power, and in-place update, and is considered as a promising candidate for main memory or data storage in the near future [38]. As reported in Figure 2.1, PCM related publications and approved US patents have grown exponentially in recent years, which indicates that PCM technology recently has renewed the interests of researcher and industry. Moreover, IBM, Samsung, Hitachi, Micron and Intel have already issued their own PCM prototype chips [3, 30, 46, 55, 66, 79, 84].

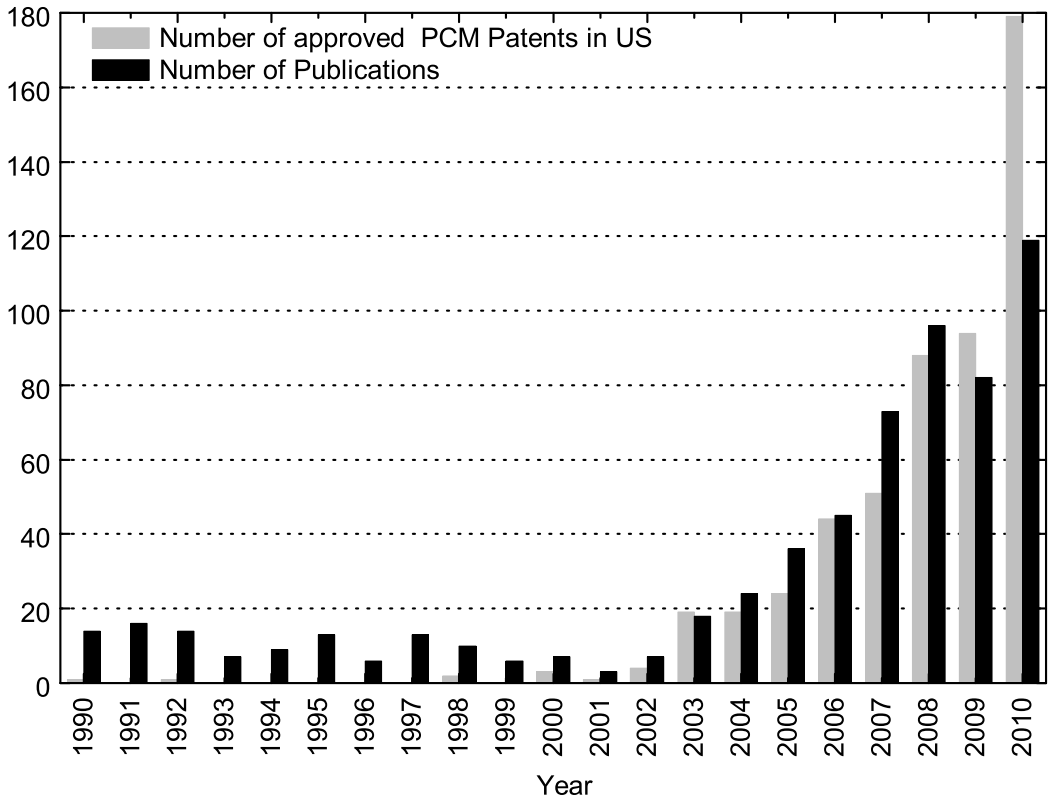


Figure 2.1. The distribution of PCM related publications and approved US patents from 1990 to 2010. The reports are collected by searching the US patents, IEEE journals and conference proceedings with the keywords: phase change memory, PCM, PCRAM and PRAM.

2.1.1 PCM Cell

Figure 2.2(a) shows a typical structure of PCM cell, which is mainly made up of the phase change material, $Ge_2Sb_2Te_5$ (GST), that is sandwiched by a top electrode and a bottom electrode. As shown, by heating up the phase change material with specific electrical pulses, it can switch between two different states, crystalline and amorphous, representing low and high resistance, respectively. Therefore, binary bit '1' or '0' can be stored in PCM cell by changing the resistance level to either of these two states. The resistance contrast between the crystalline and amorphous state is very large. For some phase change materials, it can be up to five orders of magnitude [80].

The earlier PCM prototype chips are single-level cell (SLC) PCM, in which each cell

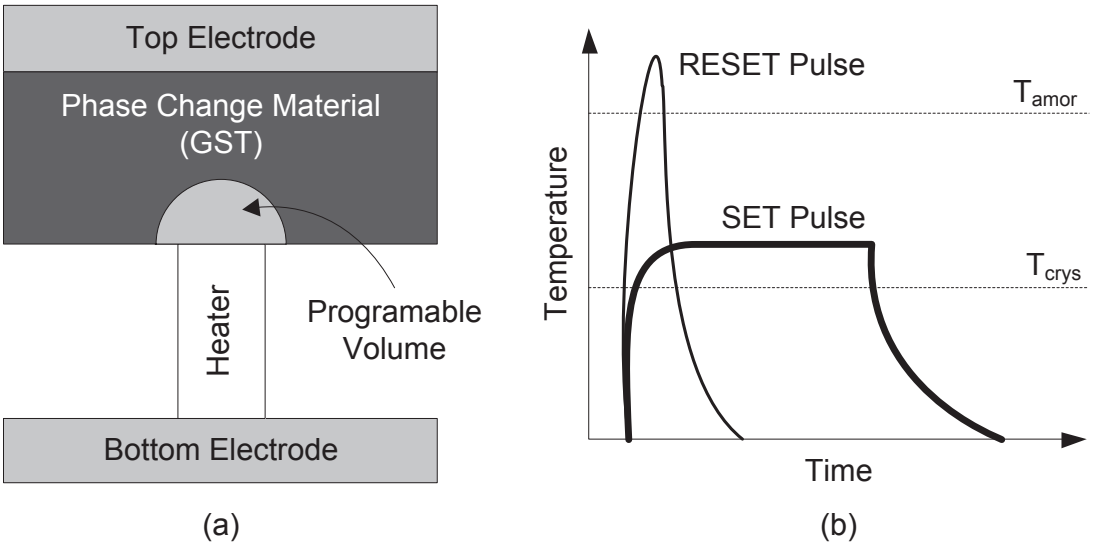


Figure 2.2. A typical PCM cell. (a) The phase change material is heated to different resistance levels by ejecting electrical current between the heater and top electrode. (b) RESET and SET pulses are performed to obtain specific levels of resistance in the PCM cell.

can only store one binary bit. However, the large resistance contrast between crystalline and amorphous states provides the opportunity to store multiple bits in one PCM cell. Recently, the multiple-level cell (MLC) PCM is proposed, wherein the phase change material is reported to achieve more intermediate resistance differences between crystalline and amorphous by carefully controlling the electrical pulses, thus multiple binary bits per cell can be represented and effectively double PCM capacity [7, 62, 76]. IBM has demonstrated a multiple-level cell (MLC) PCM [66], in which each cell can store multiple binary bits, such as '00', '01', '10', or '11'. Though we only focus on single-level cell (SLC) PCM in this thesis, the proposed methods can also be extended to MLC PCM.

2.1.2 PCM Write Operation

In order to write the data into SLC PCM cells, there are two types of write operations in PCM: RESET operation and SET operation. A RESET operation turns the phase change material into the amorphous state, while a SET operation turns the phase change material

into the crystalline state. As shown in Figure 2.2(b), to write binary '0' in a PCM cell, the RESET operation is performed to place the PCM cell into the amorphous state, by heating the phase change material above the melting temperature T_{amor} (over 600°C) with a high power but short duration pulse; To write binary '1' in a PCM cell, the SET operation is conducted to place the PCM cell into the crystalline state, by heating the phase change material above the crystallization temperature T_{crys} (over 300°C but below the melting temperature) with a moderate power but long duration pulse. On the other hand, to program the MLC PCM, an iterative programming and verifying method is employed to switch the PCM cell into four or more distinct resistance levels [23], by iteratively applying SET/RESET operation and checking whether the expected resistance is placed into a required range, such that multiple binary bits can be stored in a single PCM cell. Due to the iterative write-and-verify program method, the write operation of MLC PCM is much slower than that of SLC PCM. Moreover, the binary bits stored in PCM can be read out by sensing the resistance level of the PCM cell.

2.1.3 PCM Lifetime

As shown in Figure 2.2(b), both RESET and SET operations lead the temperature in PCM cell changes to specific levels, T_{amor} (610°C) and T_{crys} (300°C), respectively. To reach the melting temperature T_{amor}, more energy is needed for accomplishing RESET operation, which also introduces higher thermal dissipation at the same time. Therefore, repeated heat stress to PCM cells leads to the effect that a PCM cell can only sustain a limited number of RESET or SET operations. In other words, a PCM cell will be worn-out after a limited number of bit flips. For example, a single cell of Micron P5Q PCM can only sustain 10^6 write cycles [61], and the expected PCM lifetime will achieve 10^8 to 10^9 in the future [4, 38, 45]. Therefore, limited write endurance is one major challenge issue for the management of PCM-based embedded systems. To enhance PCM lifetime, an intuitive idea is through the reduction of bit flips in each PCM cell.

2.1.4 Comparison of Memory Technologies

Table 2.1 shows the characteristics of DRAM, PCM and NAND flash memory, which are mainly gathered from [25, 38]. Compared to DRAM and NAND flash memory, PCM exhibits many attractive features, such as non-volatility, bit addressability, and low standby power. As shown, the read/write latency and power consumption are slightly worse than those of DRAM, but much better than those of NAND flash memory. Thus, the distinctive advantages of PCM make it a very promising candidate for replacing DRAM-based main memory. However, the endurance of PCM is much worse than that of DRAM, so the write activities in PCM-based main memory must be considered to prohibit the PCM from being wear out faster.

Table 2.1. A Comparison of PCM with DRAM and NAND flash memory.

	DRAM	PCM	NAND Flash
Non-Volatile	NO	YES	YES
Erase Unit	Bit	Bit	Block
Software	Simple	Simple	Complex
Power	$\sim$W/GB	100-500mW/die	$\sim$100mW/die
Write Latency	$<$10ns	50-120ns	$\sim$100μs
Write Operating Voltage	2.5V	15V	3V
Read Latency	50ns	50-100ns	10-25μs
Read Operating Voltage	1.8V	$<$3V	2V
Endurance	$>10^{16}$	$10^6 - 10^8$	10^4-10^5
Retention Time	64ms	$>$10 years	$>$10 years

2.2 NAND Flash Memory

Recently, NAND flash memory is widely used as a secondary storage in embedded systems. As shown in Figure 2.3, a typical NAND flash memory is partitioned into blocks and each block is further divided into 32 or 64 pages. Each page contains 512Bytes or 2KB for data, and 32Bytes or 64Bytes for OOB (Out Of Band) area. The OOB area is primarily used to store the Error Correction Code (ECC) of the corresponding page and other information such as logical page number. There are three basic operations that can be performed on a NAND flash memory, *erase*, *write*, and *read*. A block is the smallest unit of erase operations, while a page is the minimum unit of read/write operations. In NAND flash memory, data must be written to free pages, which could lead to out of space after a number of write operations. Thus, a reclaim operation known as garbage collection [10] is invoked to regenerate free space for NAND flash memory.

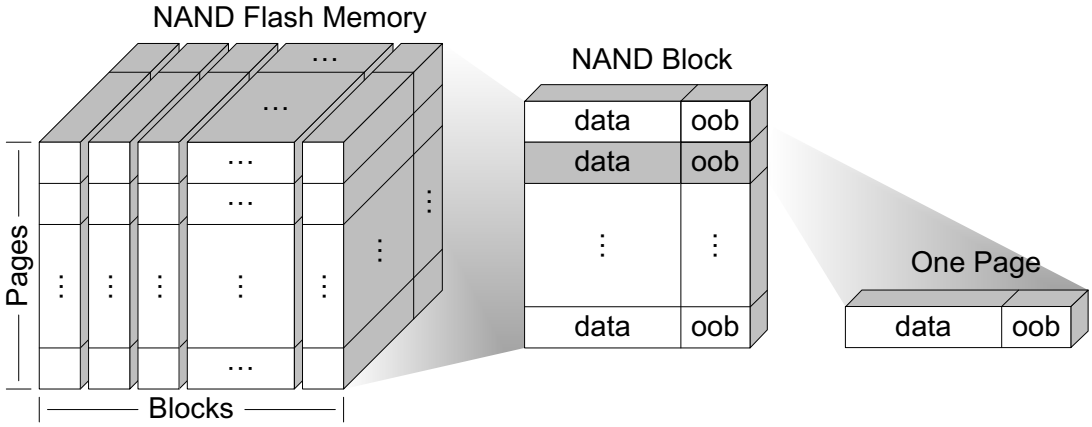


Figure 2.3. A typical structure of NAND flash memory.

As a non-volatile storage device, NAND flash memory has a lot of advantages such as small size, shock resistance and low power. However, NAND flash memory also has some constraints that impose challenges for its management. First, NAND flash memory suffers from out-of-place updates. An update (re-write) to existing data on a given physical location (known as a page) should be preceded by an erase operation on a larger region (known as a block). Second, a block has a limited erase lifetime. For example, one block

in SAMSUNG K9F1G08U0C SLC (Single-Level Cell) NAND flash has 100K erase counts, while the one in SAMSUNG K9G4G08U0A MLC (Multi-Level Cell) NAND flash has only 5K erase counts. A block becomes worn-out if its erase counts reach the limit [82]. Third, for some NAND flash memory management schemes, not all blocks in NAND flash get erased at the same rate, so the lifetime of specific blocks may decrease faster which would affect the usefulness of the entire flash memory. To overcome these constraints, it is very important to guarantee that erase or write operations be evenly distributed across all blocks (known as wear-leveling).

2.3 Flash Translation Layer

Since FTL plays an important role in NAND flash memory management, many studies for FTL have been conducted. A lot of work focuses on the address mapping of FTL [5, 6, 29, 60, 71–73, 89–92], while the other work concerns about the garbage collection [10] and wear-leveling [9, 12, 44]. According to the granularity of mapping unit, FTL designs can be mainly categorized into three types [19]: page-level mapping [5], block-level mapping [2, 6], and hybrid-level mapping [47].

As shown in Figure 2.4, in page-level FTL mapping scheme [5], each logical page number (LPN) is mapped to a corresponding physical page (PPN) in NAND flash memory. Therefore, if the file system contains n logical page units, the corresponding page-level FTL mapping table should also has n rows. As shown in the example, 16 mapping entries are allocated in the page-level FTL mapping table for the corresponding LPN to PPN mapping. With the simple mapping table, page-level FTL mapping scheme could provide efficient address translation time, less garbage collection overhead, and high space utilization but with significant memory requirement.

On the contrary, in block-level FTL mapping scheme [2, 6], one logical block (LBN) is mapped to at least one physical block (PBN) , and thus much less mapping information is needed. Figure 2.5 illustrates an example of block-level FTL mapping scheme. As shown, the logical page number 8 is divided by the number of pages in a block, to obtain the logi-

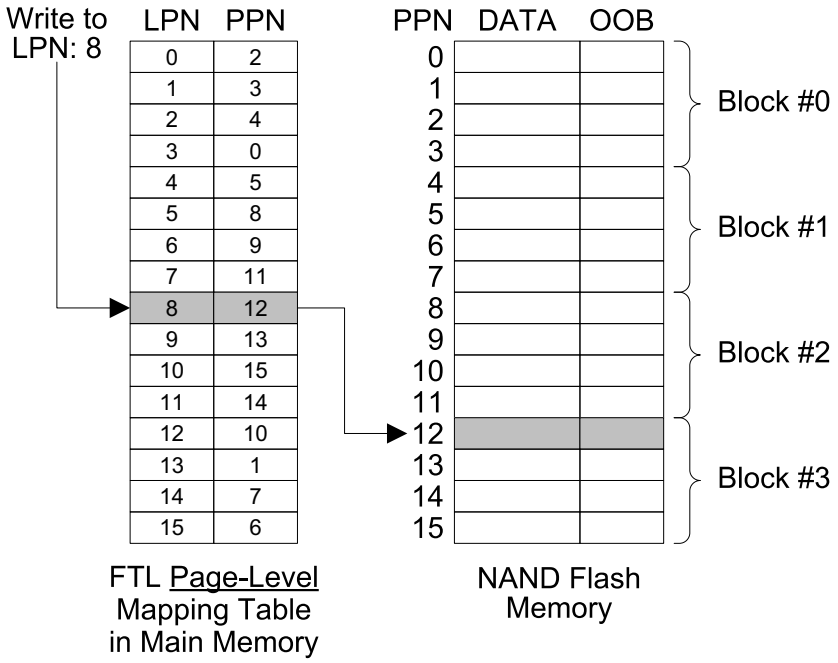


Figure 2.4. An illustration of page-level FTL mapping scheme.

cal block number 2 and the page offset 0. Then the logical block number 2 is mapped to a physical block #1 which consists of the requested page, while the page offset 0 is used as an offset to locate the page in the corresponding block #1. In block-level FTL scheme, as the number of mapping entries in block-level FTL mapping table only equals to the block number, the mapping table size can be reduced significantly. However, in block-level FTL, a logical page can only be written to a physical page with the designated page offset within a physical block. Thus, block-level FTL is not as good as page-level FTL in terms of the flexibility and performance. To achieve a trade-off between RAM cost and system performance, hybrid-level FTL mapping scheme is proposed as a compromise between page-level FTL mapping scheme and block-level FTL mapping scheme [47].

Hybrid-level FTL has been widely used, in the previous work, many studies have been conducted on hybrid-level FTL mapping schemes [17, 93, 94], especially for large-scale flash storage systems [11]. In addition, most hybrid-level FTL mapping schemes adopt log-block-based mapping mechanism for storing updates [15, 47, 54, 58, 67]. In these hybrid

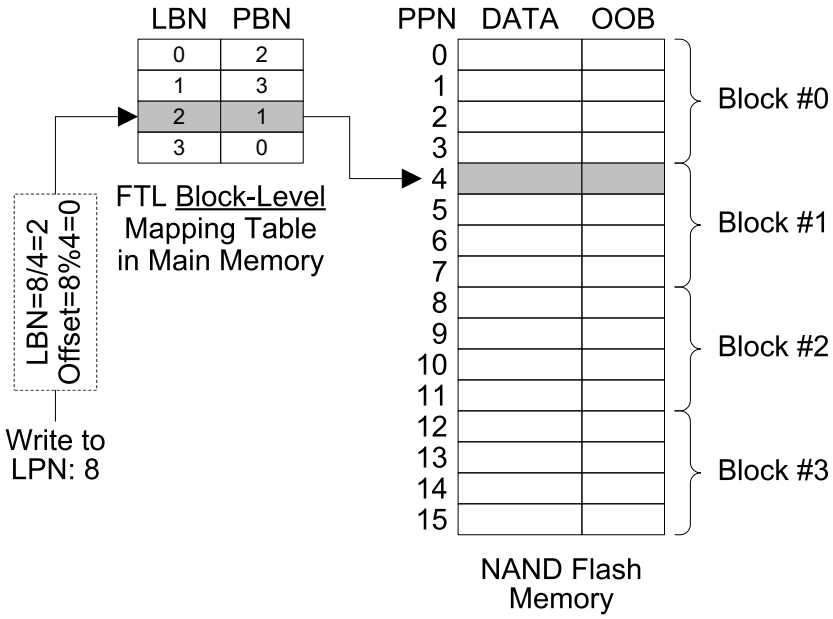


Figure 2.5. An illustration of block-Level FTL mapping scheme.

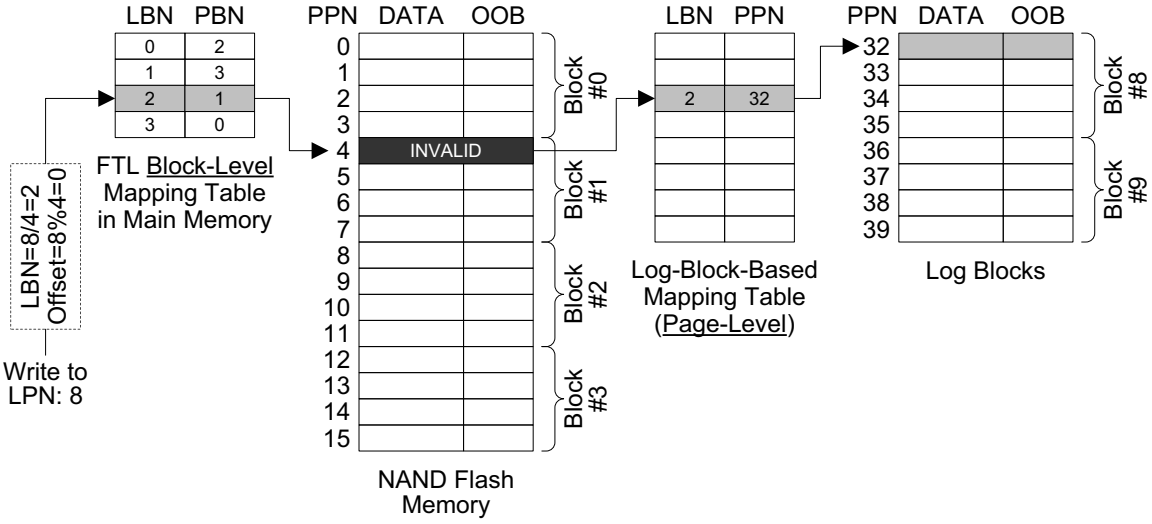


Figure 2.6. An illustration of hybrid-level FTL mapping scheme.

schemes, the blocks of NAND flash memory are divided into data blocks for new data and log blocks for the updated data. Figure 2.6 shows an example of hybrid-level FTL mapping scheme. As shown, a block-level mapping table is arranged for the mapping of data blocks,

and a page-level mapping table is used for recording the updates in the log blocks. For an update request with LBN 2 and offset 0, the fresh data is written to the first page in log block #8 instead of being stored in the original location (the first page of data block #1). Hybrid-level FTL mapping schemes have great improvement on the performance and flexibility of FTL. However, most of them need to reclaim log blocks by a merge operation which may introduce extra overhead, and the mapping table size of hybrid-level FTL mapping schemes tends to be larger than that of block-level FTL schemes. As a first step for exploring the NAND flash memory management in PCM-based embedded systems, in this thesis, we focus our work on the basic page-level and block-level FTL designs.

2.4 Summary

In this chapter, we introduced the background knowledge of phase change memory. Then we briefly discussed the characteristics of NAND flash memory. Finally, we presented different types of flash translation layers that are used for managing NAND flash memory in embedded systems.

CHAPTER 3

WAP-FTL: A PAGE-LEVEL PCM-AWARED FLASH MEMORY MANAGEMENT TECHNIQUE

3.1 Overview

As an emerging non-volatile memory, phase change memory (PCM) exhibits its potential of being incorporated into current memory hierarchies, and several studies so far show that the advantages of PCM make it an ideal replacement for DRAM in main memory [21, 26, 52, 53, 74, 77, 78, 101]. Unfortunately, compared to DRAM, PCM suffers from a limited number of write cycles, i.e., a cell of PCM can only sustain 10^6 to 10^8 writes before worn-out [38], which imposes challenge for it to store frequently updated data. For example, Micron Omneo^{TM}P5Q PCM has only 10^6 write cycles per cell [61]. As main memory is a heavily accessed component, it is therefore important to reduce write activities in PCM for improving the lifetime of PCM-based main memory. Several techniques recently have been developed to reduce write activities in PCM at architecture/software level [16, 26, 27, 33, 34, 52, 74, 88, 101].

Most of the previous work provide good solution and improve the performance. However, none of them targets at the emerging PCM-based embedded systems, wherein PCM is used as main memory while NAND flash memory is used as a secondary storage. The management process of NAND flash memory could impose significant write activities in PCM-based main memory, and as a result degrade the PCM lifetime. Therefore, unlike previous work, we target at the hybrid embedded systems with PCM and NAND flash memory, and propose a write-activity-aware page-level flash memory management technique to

reduce write activities for PCM, such that the lifetime of PCM-based embedded systems is enhanced.

As shown in Figure 1.2, to access NAND flash memory in the PCM-based embedded systems, flash translation layer (FTL) is adopted to emulate NAND flash memory as a disk drive by mapping logical addresses to physical addresses in NAND flash memory at a granularity of page-level or block-level [37, 51]. Among the proposed FTL schemes [5, 6, 18, 19, 29, 93], FTL mapping table is employed to keep track of the continually updated mapping records, in terms of the I/O requests. 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. However, the most-updated FTL mapping table can be lost due to power failure in DRAM-based main memory. In PCM-based embedded systems, to utilize the non-volatile feature of PCM, FTL mapping table can be kept into PCM-based main memory permanently without considering power failure.

Recently, Kim et al. [48] targets at the PCM-based embedded systems, and propose a page-level FTL, namely *h*FTL, in which page-level FTL mapping table is kept in PCM and user data is stored in NAND flash memory. Nevertheless, *h*FTL does not consider redundant write activities occurred in PCM because of the frequently updated FTL mapping table, which may lead to a shortened PCM lifetime. As the lifetime of PCM is mainly determined by the maximum number of bit flips in each PCM cell, it is important to reduce the maximum number of bit flips in each PCM cell to enhance the reliability of the entire system. New techniques, therefore, are needed to reduce unnecessary write activities that may increase the maximum number of bit flips in PCM due to the update process of FTL mapping table, such that the lifetime of the PCM-based embedded systems is enhanced.

In this chapter, we propose a **W**rite-**A**ctivity-aware **P**age-level **FTL** technique, called **WAP-FTL**, to reduce write activities in PCM during the management process of NAND flash memory and, at the same time, to enhance the lifetime of the PCM-based embedded systems, with the advantage that no changes are required to the file system, and hardware implementation of the NAND/PCM chip. Note that mapping records inside FTL mapping

table are represented in a binary form in PCM. Different from the redundant write avoidance method mentioned by most of the previous work [27,33,43,101], we employ a write-activity-aware strategy in flash translation layer. Our basic idea is to preserve each bit in page-level FTL mapping table hosted by PCM, 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. Once a write request arrives in NAND flash memory, unlike the traditional page-level FTL scheme [5, 48], the proposed page-level FTL technique, WAP-FTL can actively choose a physical page whose physical address could effect the minimum number of bit flips in FTL page-level mapping table, so as to effectively reduce write activities on PCM cells.

We conduct a series of experiments on a set of realistic I/O traces collected from notebook and Google Android platform. We applied our write-activity-aware strategy to a representative technique *h*FTL [48], and compared with *h*FTL in terms of the total and maximum number of bit flips in each PCM cell with various configurations. The experimental results show that the proposed technique can effectively reduce write activities in PCM, however, the number of bit flips in PCM will increase after garbage collection happens. We analyze that WAP-FTL does not consider the access behavior of I/O requests, which consists of random and sequential requests, and the proposed write-activity-aware strategy introduces extra overhead, i.e., valid page copy during garbage collection leads to significant bit flips in FTL mapping table.

This chapter makes the following contributions:

- We present for the first time a write-activity-aware page-level flash memory management technique to reduce write activities in PCM-based embedded systems for enhancing the PCM lifetime.
- We demonstrate the limitation of the proposed page-level flash memory management by comparing with representative technique using a set of realistic I/O workloads collected from notebook by DiskMon [1].

The rest of this chapter is organized as follows. Section 3.2 introduces the PCM-based embedded systems, and the representative FTL implementations for hybrid architecture. Section 3.3 presents our WAP-FTL technique. Section 3.4 presents the experimental results. Section 3.5 concludes the chapter.

3.2 Background and Motivation

In this section, we first introduce the targeted PCM-based embedded systems, and then describe the motivational example. Finally, we present the motivation of this chapter.

3.2.1 PCM-based Embedded Systems

In this chapter, we target at the PCM-based embedded system with page-level flash memory management scheme. As shown in Figure 3.1, a typical PCM-based embedded system consists of a PCM-based main memory and a NAND flash memory. PCM acts as a main memory for maintaining frequently accessed OS pages and the FTL mapping table, and NAND flash memory is employed as a secondary storage for storing user data that are accessed by file systems.

Thanks to the page-level flash translation layer, write requests from file systems to NAND flash memory is handled transparently, and the mapping from logical address (logical page number) to physical address (physical page number) will be updated continually in FTL mapping table hosted by PCM. As influenced by the I/O requests, the page-level FTL mapping table is the most heavily updated component in PCM and may shorten PCM lifetime. To avoid the lifetime degradation of PCM, it is necessary to make the page-level FTL scheme write activity aware in the PCM-based embedded system. Therefore, in this chapter, we propose a write-activity-aware page-level FTL scheme, WAP-FTL, for reducing write activities in the PCM-based embedded systems.

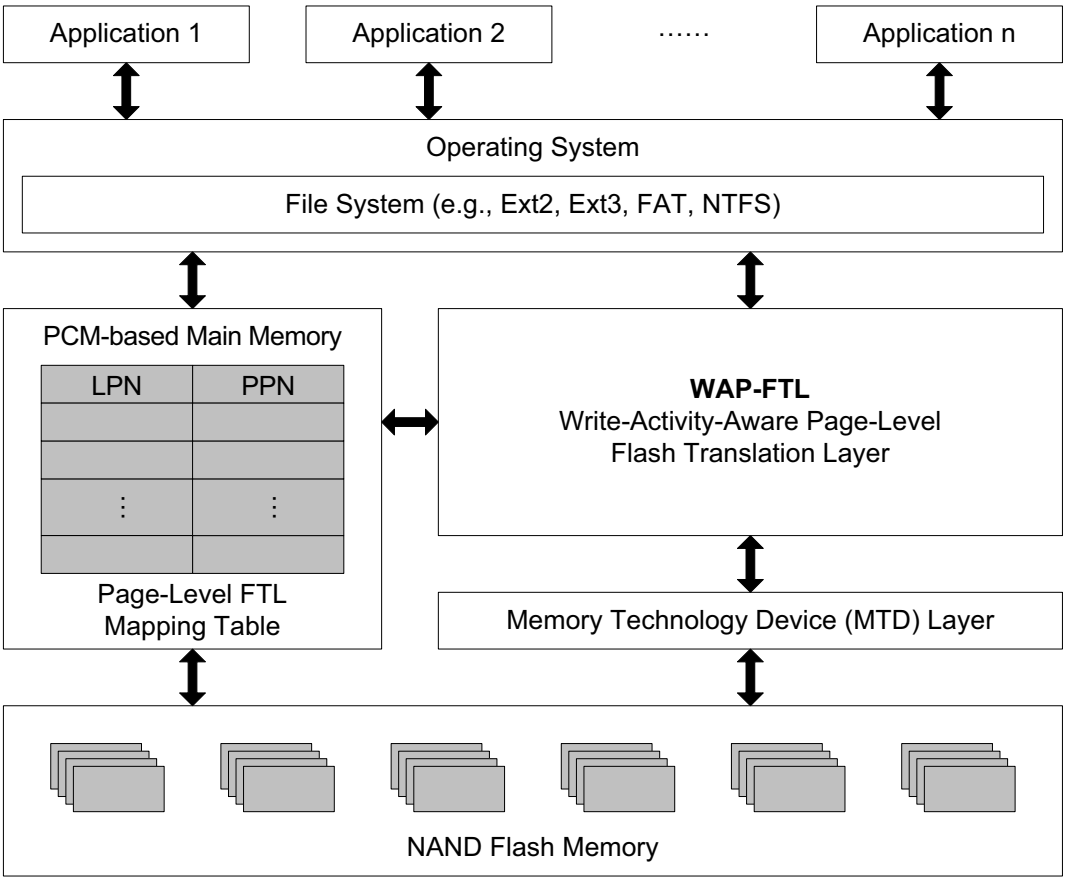


Figure 3.1. PCM-based embedded systems with the proposed write-activity-aware page-level WAP-FTL technique.

3.2.2 Motivational Example

In this section, we briefly revisit the *h*FTL scheme which is currently the only one FTL scheme proposed for managing NAND flash memory in PCM-based embedded systems [48].

*h*FTL is based on page-level mapping scheme [5], 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, *h*FTL 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

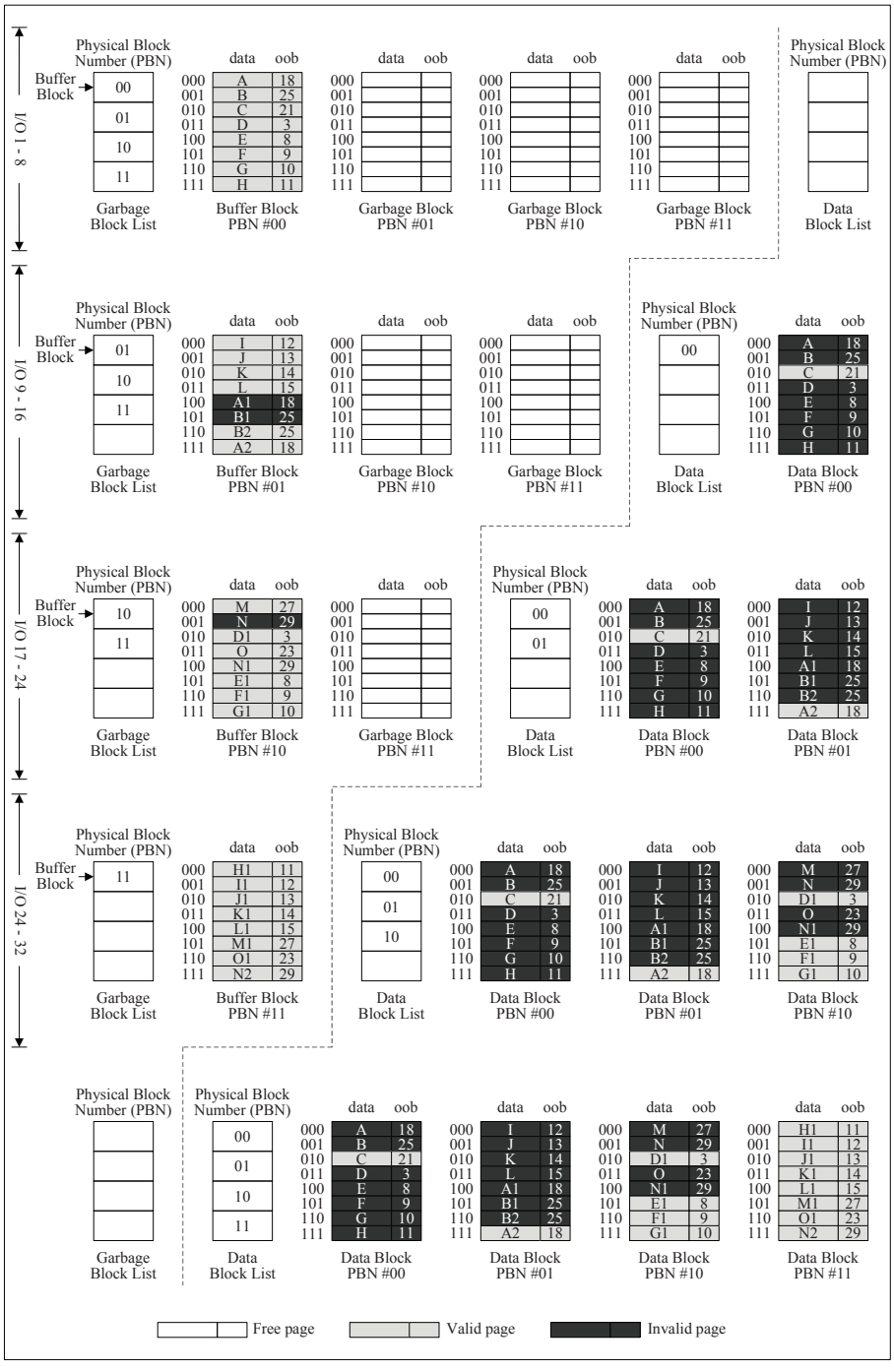
of garbage blocks, a garbage collection operation is performed to reclaim a block from the data blocks. In *h*FTL, a page-level mapping table in PCM keeps track of mappings between logical page number (LPN) and physical page number (PPN), in terms of the I/O requests. Consequently, the mapping table is updated frequently and thus imposes the endurance issue for PCM. A motivational example is illustrated in Figure 3.2.

In the example, 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 *h*FTL and our PCM-FTL scheme, the physical page number (PPN), physical block number (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 Figure 3.2(a). According to the given I/O requests, the status variation of the blocks in NAND flash memory is shown in Figure 3.2(b). For *h*FTL, 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.

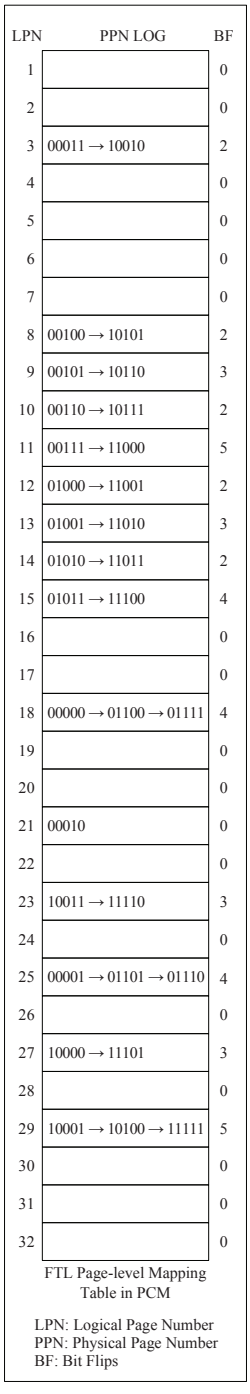
As shown, the first request is written to LPN (#18). A new buffer block (PBN #00) is allocated from the garbage block list, and the content A with the corresponding LPN (#18) are stored in the first page of current buffer block (PBN #00). Meanwhile, the mapping information of LPN (#18) and PPN (#00000) is stored into the mapping table shown in Figure 3.2(c). 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) are 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.

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
LPN	18	25	21	3	8	9	10	11	12	13	14	15	18	25	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

(a)



(b)



(c)

Figure 3.2. Motivational example. (a) I/O access sequence. (b) The status variation of blocks in NAND flash memory. (c) The status variation of FTL page-level mapping table in PCM.

3.2.3 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 mapping information in the mapping table is updated as well. In Figure 3.2(c), 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 entries have the maximum number of bit flips 5. Since PCM cell can only sustain limited number of write cycles, frequent update operations in mapping table will lead to the fast worn out of PCM. In addition, *h*FTL is further evaluated with a variety of realistic I/O traces in Section 3.4, and a part of the evaluation results is shown in Figure 3.3. As shown, the maximum number of bit flips in PCM cells from almost half of the traces achieves the PCM endurance limit, if a PCM cell can only sustain 10^6 write operations (e.g., Micro P5Q PCM [61]). Besides, the total number of bit flips in PCM cells is significantly higher, which can also influence the power efficiency of PCM. 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 PCM-based embedded systems.

As mentioned above, several hardware optimization techniques for PCM have been proposed [52, 101], 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, the technique proposed in this chapter could actively choose mapping information (e.g., PBN) which is almost the same as the mapping to be updated in the page-level FTL mapping table, such that the number of write activities in PCM is reduced.

3.3 WAP-FTL: PCM-Awared Page-Level FTL

In this section, we introduce our WAP-FTL, a write-activity-aware page-level FTL, that can reduce write activities in PCM. We first present an overview of WAP-FTL in Section 3.3.1. We then provide a detailed description of WAP-FTL in Section 3.3.3.

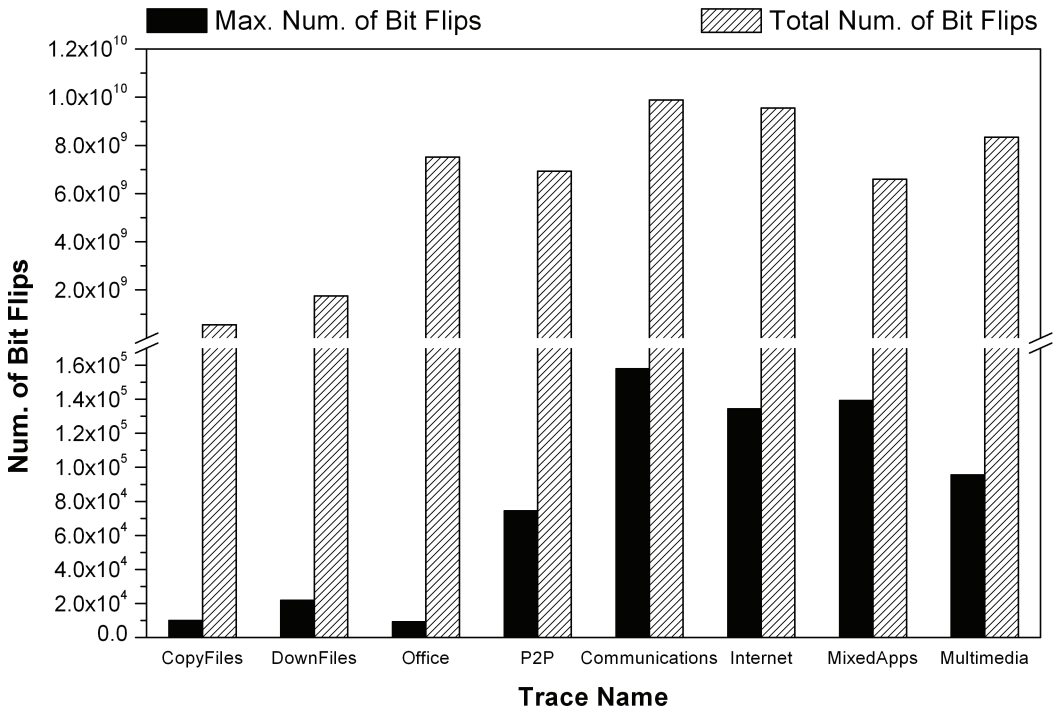


Figure 3.3. The evaluation results of *h*FTL in terms of the maximum and total number of bit flips in PCM cells over different I/O traces.

3.3.1 Overview of WAP-FTL

The objective of WAP-FTL is to reduce write activities in PCM-based embedded systems. Therefore, the basic idea of WAP-FTL is to preserve each bit in FTL mapping table hosted by PCM from being inverted frequently, e.g., $0 \rightarrow 1 \rightarrow 0$, during the update process of FTL mapping table, such that the maximum number of bit flips in each PCM cell is reduced and the PCM lifetime is enhanced. Thus, in WAP-FTL, we develop a write-activity-aware strategy technique, which could actively find physical pages for write requests issued into NAND flash memory. The reason behind this idea is that the physical page number of the physical page should be updated in the page-level FTL mapping table in PCM following the write requests, thus write activity reduction can be obtained by carefully select the physical page whose page number incurs the minimum number of bit flips in PCM cells.

3.3.2 Write-Activity-Aware Strategy

Figure 3.4 shows the proposed write-activity-aware strategy. As shown, for the write requests, the traditional page-level FTL scheme, e.g., *h*FTL, allocates physical pages consecutively and updates mapping table with the corresponding physical page number without considering write activities. In this example, *h*FTL selects the first available physical page and introduces 5 bit flips in PCM. However, unlike the traditional page-level FTL, our write-activity-aware strategy first checks the corresponding mapping record in FTL mapping table ('0111011011011'), and then actively selects an available physical page whose physical page number has the minimum Hamming distance with the original mapping record. In this figure, a physical page '0111011010011' is found since it incurs only 1 bit flip in PCM during the update process of FTL mapping table. It can be seen that the proposed write-activity-aware strategy can effectively reduce the number of bit flips in PCM.

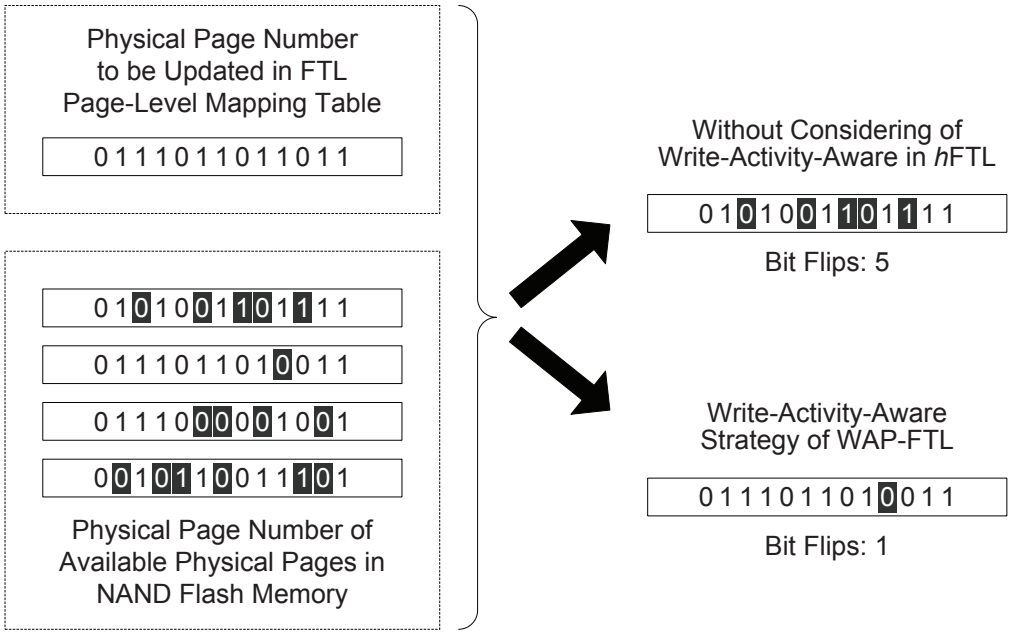


Figure 3.4. The write-activity-aware strategy of WAP-FTL.

3.3.3 WAP-FTL Description

In WAP-FTL, based on the write-activity-aware strategy, we actively choose a physical page according to the write requests, and the PPN of the selected physical page must cause the minimum number of bit flips when comparing with the original PPN in the mapping table. By applying WAP-FTL, the number of bit flips is reduced when the page mapping table is updated. Consequently, the endurance of PCM can be enhanced.

Algorithm 3.3.1 shows how WAP-FTL actively finds a particular physical page whose PPN incurs minimum number of bit flips in the FTL page-level mapping table. As shown, when an write request arrives, we first get the old PPN and old PBN (lines 1-2). If the corresponding entry in the mapping table is empty, i.e., the old PPN is NULL, then we randomly find a block with free pages and use its PBN as a temporary PBN (Tmp_PBN) for further checking (lines 4-5); Otherwise, the LPN has already been mapped, i.e., the old PPN is not NULL, then assign the old PBN to the temporary PBN, and invalid the old content in the page #Old_PPN (lines 6-8). Then get the PPN of the current free page of the block #Tmp_PBN, and assign it to Tmp_PPN (line 9). If the bit flips (BF) between Tmp_PPN and Old_PPN equals to 0 or 1, which means that we find a physical page whose PPN causes the minimum number of bit flips in the FTL page-level mapping table, thus we get the new PPN and update the FTL page-level mapping table (lines 11-12, line 24). However, if the BF is greater than 1, then change one bit in Tmp_PBN, so we can get the other PBNs, and thus the block #Tmp_PBN and the blocks whose PBN with one bit difference with #Tmp_PBN forms a current block set (lines 13-14). In the current block set, we find a physical page whose PPN causes the minimum number of bit flips when comparing with Old_PPN (lines 15-16). If a physical page is found, then get the new PPN and update the FTL page-level mapping table (lines 17-18, line 24); Otherwise, a physical page is selected from a block not in the current block set, then get the new PPN and update the mapping table (lines 19-20, line 24).

An example of WAP-FTL is shown in Figure 3.5. This example is based on the access sequence and the assumptions of NAND flash memory for the motivational example shown in Figure 3.2. As shown, for the 4th request with LPN (#3), we random assign a

Algorithm 3.3.1 The algorithm of WAP-FTL

Input: Logical Page Number (LPN), page number of a block (BlkPgNum).

Output: New Physical Page Number (New_PPN).

```
1: Old_PPN = MapTab[LPN].
2: Old_PBN = Old_PPN / BlkPgNum.
3: if Old_PPN == NULL then
4:     Tmp_PBN ← randomly find a block with free pages.
5: else
6:     Tmp_PBN = Old_PBN.
7:     Invalid the old content in the page #Old_PPN.
8: end if
9: Tmp_PPN ← PPN of the current free page in block #Tmp_PBN.
10: BF ← Bit Flips between Tmp_PPN and Old_PPN.
11: if BF equals 0 or 1 then
12:     New_PPN = Tmp_PPN.
13: else
14:     The block #Tmp_PBN, and the blocks whose PBN has only one bit difference with Tmp_PBN forms a
        current block set.
15:     for all blocks in current block set do
16:         Find a physical page whose PPN incurs the minimum number of bit flips when comparing with
            Old_PPN.
17:         if Success then
18:             New_PPN ← PPN of the selected physical page.
19:         else
20:             New_PPN ← PPN of the page in a block not in the current block set.
21:         end if
22:     end for
23: end if
24: Return New_PPN, write the new content into the page #New_PPN and update the mapping table with
    New_PPN.
```

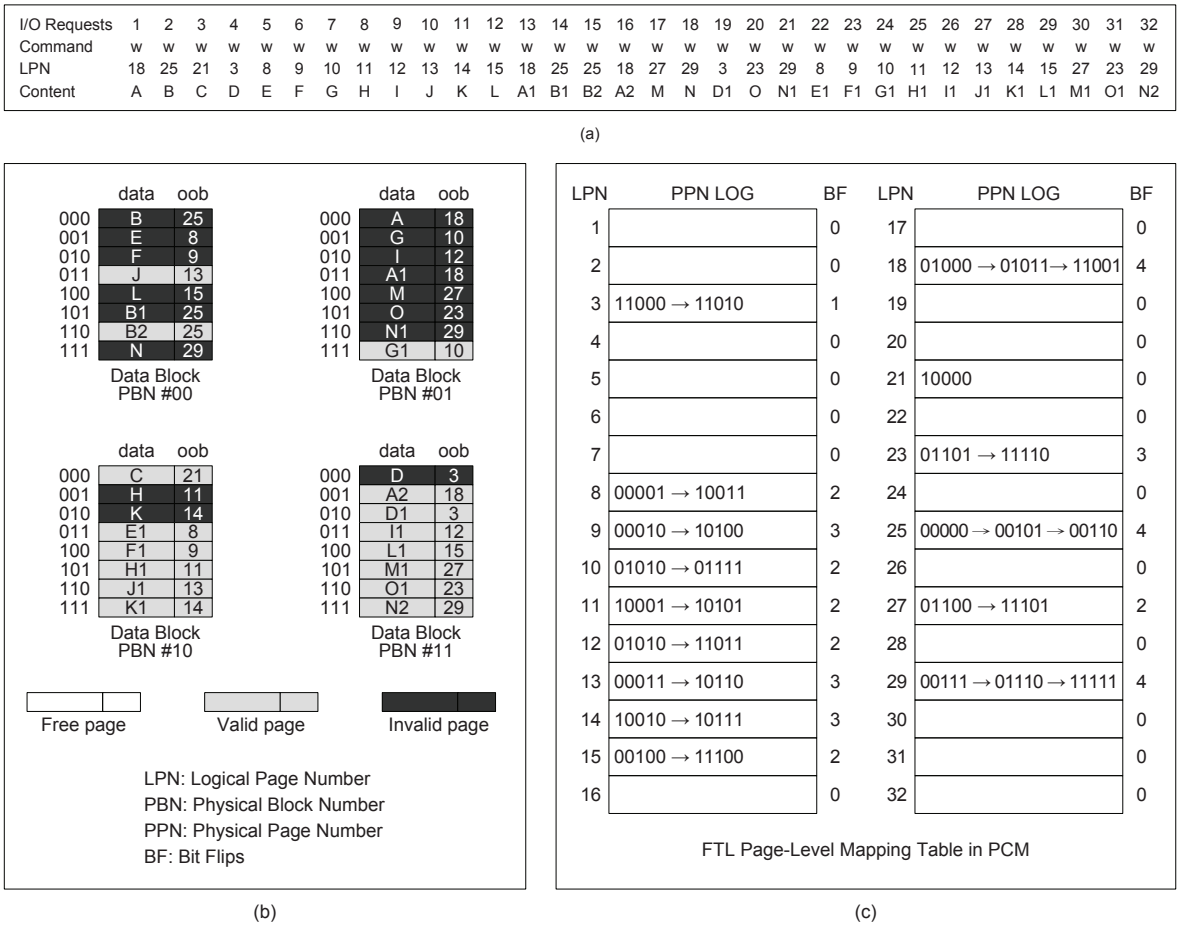


Figure 3.5. An example of WAP-FTL. (a) I/O access sequence used by the motivational example in Figure 3.2. (b) The status variation of blocks in NAND flash memory. (c) The status variation of FTL page-level mapping table in PCM.

block (PBN #11) since the corresponding entry for this LPN is empty, and the content D is written into the current free page (#11000) of block (PBN #11). For this request, there is no bit flip when updating the FTL page-level mapping table in PCM. It can be seen that D is updated by a new content $D1$ in the 19th request, and $D1$ is written into the physical page (#11010) according to our write activity-aware strategy. When the 19th request arrives, we use the LPN (#3) to get the old PPN (#11000) from the mapping table. Then we know the old content D of this LPN (#3) is stored in the page PPN (#11000), thus this page is set invalid. According to the old PPN (#11000), the corresponding PBN can be obtained (#11), and the PPN of the current free page in this block is #11010. Next, comparing #11010 with

the old PPN #11000, the number of bit flips is only 1. Therefore, in terms of our algorithm, we do not need to find other physical pages, the new content $D1$ can be directly written into the physical page (#11010). By using WAP-FTL, we only need to write one reversed bit in the mapping table for this update operation. By contrast, two reversed bits have to be written into the mapping table according to *h*FTL. After processing all requests, we found that the total number of bit flips are 37 by our WAP-FTL, while the total number of bit flips in PCM is 44 by *h*FTL. Our scheme achieves a reduction of 15% in total number of bit flips, which confirms that our write-activity-aware strategy can effectively reduce write activities in PCM. The experimental results in Section 3.4 also show that our scheme can effectively reduce the total number of bit flips.

3.4 Evaluation

To evaluate the proposed WAP-FTL scheme, we conducted a series of experiments and present the experimental results with analysis in this section. Below, we first introduce the experimental setup. Then, based on our simulation framework, we present the experimental results and discussion for WAP-FTL. Finally, we analyze extra overhead caused by the write-activity-aware strategy adopted by WAP-FTL.

In this chapter, we assume that the FTL page-level mapping table is 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.

3.4.1 Experimental Setup

Table 3.1 summarizes the setup configurations of our evaluation. The evaluation is conducted through a trace-driven simulation framework, in which a simulator is designed to evaluate WAP-FTL and *h*FTL using a variety of realistic I/O traces collected from notebook and Android platform. The I/O traces reflect the realistic workloads of the system in accessing the secondary storage for daily use.

Table 3.1. Experimental Setup.

Notebook Configuration	CPU	Intel Dual Core 2GHz
	Disk Space	200GB
	RAM	2GB
	DiskMon Traces	CopyFiles, DownFiles Office, P2P
Android Emulator Configuration	CPU	ARM926EJ-S
	OS Kernel	Linux 2.6.29
	I/O Scheduler	NOOP
	Android Version	2.3
	Android Traces	Communications, Internet MixedApps, Multimedia
Simulation Environment	OS Kernel	Linux 3.0
	Flash Size	1GB & 4GB
	PCM Size	128Mb

In the notebook platform with an Intel Dual Core 2GHz processor, a 200GB hard disk, and a 2GB DRAM, the I/O traces of four applications frequently used in daily life are gathered by DiskMon [1]. 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. For each trace, to map the address of the disk space into the address of the NAND flash space, we adopt a module operation, i.e., section number in HDD address mod total page number of NAND flash. The statistics of each trace are listed below:

- Copyfiles: I/O requests - 13608439, write ratio - 78%

- DownFiles: I/O requests - 26588711, write ratio - 72%
- Office: I/O requests - 85297213, write ratio - 77%
- P2P: I/O requests - 251789825, write ratio - 29%

Besides, to fully evaluate the proposed FTL scheme in the embedded environment, some Google Android traces are collected from Google Android™2.3 with Android Emulator (included in Android SDK). We modified the Linux kernel shipped with Android to record I/O requests in system log. *NOOP* scheduler in Linux is selected as NAND flash memory is truly random-access and does not need optimization for "seeking" operations found in traditional hard disks. Traces are gathered by *Android Debug Bridge* in Android SDK from the emulator to host computer. In order to reveal the actual impacts of the experimental schemes, we collected traces under heavy-loaded environment by using *Monkey*, which is a automatic stress test tool provided by Android SDK. With applications specified, it generates random events for them and send the events to the emulator for execution. Four traces are collected, as shown in Table 3.2. Each trace is collected by running the applications specified by the second column in Table 3.2. *Internet* focuses on online activities; *Multimedia* consists of a set of frequently used multimedia applications; *Communication* includes applications that will be frequently used when the user is trying to connect the rest of the world. In order to represent the randomness of user behavior, we collected *MixedApps*, which are both mixes of applications across different application domains.

Table 3.2. Android Trace Applications

Trace	Applications
Internet	Web Browser, EMail, Search, Settings
Multimedia	Music, Camera, Gallery, Settings
Communications	Phone, Contacts, Messaging, Voice Dialer
MixedApps	Browser, EMail, Music, Contacts, Settings

The simulation framework of our simulation platform is shown in Figure 3.6. This simulation framework simulates our WAP-FTL scheme over the PCM-based embedded systems, which consists of a NAND flash memory and a PCM for storing page-level FTL mapping table. In our experiments, the traces along with various parameters of NAND flash memory, such as block size, page size, etc, are fed into our simulator. The page size, number of pages in a block, and size of the OOB for each page are set as 2KB, 64, and 64 Bytes, respectively. To fully evaluate our technique, we conduct the experiments on a PCM-based embedded system with 1GB and 4GB NAND flash memory, respectively.

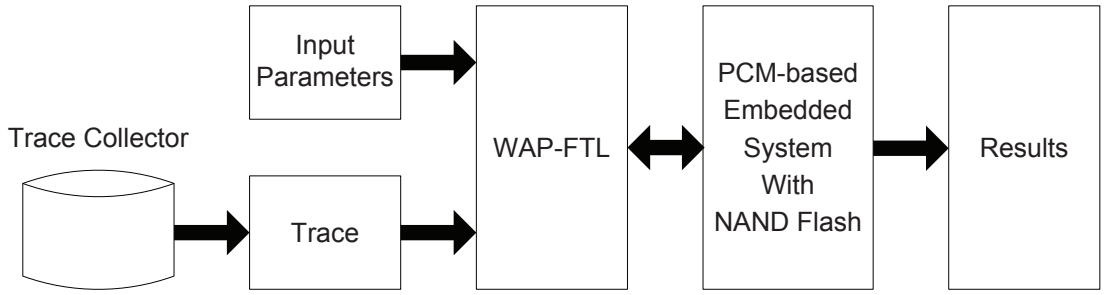


Figure 3.6. The framework of simulation platform for evaluating the proposed WAP-FTL technique.

3.4.2 Results and Discussion

In this section, we compare and evaluate our proposed WAP-FTL technique over the representative page-level FTL scheme, *h*FTL [48] in terms of two performance metrics: the maximum and total number of bit flips in PCM. Experiments are conducted based on our PCM-based embedded system simulator with 1GB and 4GB NAND flash memory over eight distinct traces gathered from notebook and Google Android^{TM}platform.

PCM Endurance

Table 3.3 and Table 3.4 show the experimental results for the maximum and total number of bit flips of our proposed scheme WAP-FTL and the representative page-level FTL scheme

Table 3.3. WAP-FTL versus *h*FTL in terms of the maximum number of bit flips in PCM cells. (1GB NAND flash memory)

Trace Name	*h*FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	188	325	326	326	328	9977
DownFiles	106	186	187	188	190	21945
Office	12	13	14	41	43	9385
P2P	35	118	194	228	276	74540
Communications	88	164	254	340	402	158029
Internet	70	151	231	319	378	134281
MixedApps	93	183	267	352	441	139241
Multimedia	93	151	154	155	156	95621
	WAP-FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	134	239	239	239	239	14209
DownFiles	75	111	111	112	112	135775
Office	7	9	10	25	25	163538
P2P	25	65	115	127	155	261351
Communications	77	126	183	232	259	157907
Internet	64	121	167	215	243	134263
MixedApps	82	139	183	249	297	139243
Multimedia	87	128	128	128	128	95632

Table 3.4. WAP-FTL versus *h*FTL in terms of the total number of bit flips in PCM cells. (1GB NAND flash memory)

Trace Name	*h*FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	983057	2174684	3303546	4583374	5857110	559496658
DownFiles	1003343	2105436	3210454	4524334	5837597	1756464372
Office	985871	2085896	3276887	4539453	5842018	7520028995
P2P	992452	2115462	3334764	4647266	5970250	6929967624
Communications	1002453	2074710	3118356	4178193	5228922	9883730134
Internet	1031781	2077305	3124603	4139189	5186971	9555187107
MixedApps	977832	2013718	3055599	4093552	5119704	6591084832
Multimedia	982136	2046396	3083667	4111219	5153549	8347917692
	WAP-FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	622096	1058139	1976801	2937719	3765586	1324170182
DownFiles	708192	1209301	1722262	2226636	3327637	4917431792
Office	697521	1479747	2156464	2655455	3255932	23639132918
P2P	765204	1479963	2118783	2777092	3376001	22401489990
Communications	202182	384743	562976	737703	913180	4810126773
Internet	213625	392223	581538	751353	931305	4650348064
MixedApps	198253	379117	556384	728511	918594	3159769242
Multimedia	200650	476399	689614	862226	1069147	4301343004

Table 3.5. WAP-FTL versus *h*FTL in terms of the maximum number of bit flips in PCM cells. (4GB NAND flash memory)

Trace Name	*h*FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	323	328	358	1208	1659	10461
DownFiles	188	196	466	1753	2135	20857
Office	41	58	69	72	76	36667
P2P	228	723	1110	1308	1339	86383
Communications	342	411	457	537	670	158029
Internet	324	452	507	567	698	134281
MixedApps	354	497	513	544	817	149439
Multimedia	155	290	433	436	499	95846
	WAP-FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	347	349	369	1244	1650	10880
DownFiles	191	191	423	1527	1837	36920
Office	23	30	37	38	39	172509
P2P	145	515	831	969	985	228553
Communications	318	383	459	520	567	157541
Internet	313	417	482	525	572	134125
MixedApps	359	470	478	535	687	138895
Multimedia	161	293	414	415	425	95510

Table 3.6. WAP-FTL versus *h*FTL in terms of the total number of bit flips in PCM cells.

(4GB NAND flash memory)

Trace Name	*h*FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	4494710	9327633	14815454	20314533	25833983	122605530
DownFiles	4357898	9200422	14593255	19767278	25266362	842401436
Office	4435404	9558630	15000280	20751651	26342930	6981790260
P2P	4424885	9439670	14574473	20019491	25736434	15269865958
Communications	4214731	8444464	12641676	16788274	21043933	10305765684
Internet	4166898	8363191	12509433	16635517	20747378	9988536600
MixedApps	4129671	8313322	12505355	16723162	20913638	5130339159
Multimedia	4144192	8417270	12728467	17077552	21224068	8654772875
	WAP-FTL					
	% of page usage in NAND flash memory					Finish
	20%	40%	60%	80%	100%	
CopyFiles	3086425	6958867	8872671	10392954	11917754	156287827
DownFiles	3452515	7227272	9498394	11215494	12852766	1209891021
Office	3214941	5533978	7811282	9724357	11747696	12500791348
P2P	3269436	6406322	9266073	11719374	14284551	22148554910
Communications	694663	1338881	2001803	2644835	3352860	4801121525
Internet	702007	1344439	1996640	2641505	3346537	4640901865
MixedApps	684108	1327512	1978781	2622322	3312456	3150978018
Multimedia	769244	1435469	2122868	2799786	3494992	4290735847

*h*FTL, over a simulator of PCM-based embedded system with 1GB NAND flash memory, respectively. In each table, columns 2-6 present the maximum/total number of bit flips in PCM cells when a specific percentage of page usage in NAND flash memory is achieved. And the last column of each table present the maximum/total number of bit flips in PCM cells when all write requests of a trace are served. In Table 3.3, we observe that WAP-FTL reduces the maximum number of bit flips a lot than *h*FTL before all pages are allocated for write requests over different traces. However, for the last column, we find that the maximum number of bit flips obtained by WAP-FTL is almost the same as that of *h*FTL, or even worse than that of *h*FTL (e.g., DownFiles and P2P), after all write requests of a trace are served over different traces. This means that the write-activity-aware strategy in WAP-FTL cannot always effectively reduce write activities in PCM cells. The similar scenario is also found in Table 3.4. As shown, though WAP-FTL can significantly reduce the total number of bit flips before all pages are full, it cannot reduce activities in PCM finally for all traces.

To verify the above observation, we further extend the volume of NAND flash memory to 4GB in our simulator and evaluate our technique. Table 3.5 and Table 3.6 report the similar results for the maximum and total number of bit flips in PCM cells obtained by WAP-FTL and *h*FTL, respectively. Therefore, these results demonstrate that WAP-FTL may not be a better solution for reducing write activities in PCM-based embedded systems. The reason of the above scenarios is caused by extra valid page copy during garbage collection in our write-activity-aware strategy.

Overhead

As mentioned before, to reduce write activities during the update process of FTL mapping table, our scheme will actively find an available physical page whose page number incurs the minimum number of bit flips in PCM. Therefore, unlike the traditional page-level FTL scheme, such as *h*FTL, which allocates physical pages consecutively for incoming write requests, our WAP-FTL actively selects available physical page among all blocks. Though our method could reduce write activities in PCM at an earlier stage of I/O requests. However,

for the later sequential update requests, it will invalidate some updated pages across multiple blocks that also contain some valid pages. Once these blocks are full and garbage collection is triggered to reclaim free space, the valid pages in these blocks should be copied into some free blocks and their mapping records in the mapping table are updated accordingly. Thus, extra bit flips are induced during the valid page copy in garbage collection. In contrast, in *h*FTL, all pages in a block may be invalidated consecutively following the sequential update requests, and thus no extra page copy is needed during the garbage collection. So it can be seen that the results of WAP-FTL are similar to or worse than that of *h*FTL once all pages are used in NAND flash memory.

3.5 Summary

In this chapter, we have proposed a write-activity-aware page-level flash memory management scheme, named WAP-FTL, to exploit the advantages of the well-known FTL implementations in order to reduce write activities in PCM for enhancing lifetime of the PCM-based embedded systems. In our WAP-FTL, the write activity reduction is achieved by preserving each bit in page-level FTL mapping table that hosted by PCM from being inverted frequently. Unlike the traditional page-level FTL scheme [48], WAP-FTL can actively choose a physical page whose physical address incurs the minimum number of bit flips in FTL page-level mapping table hosted by PCM, so as to effectively reduce write activities in PCM cells. However, with a set of real-life workloads, the experimental results show that our WAP-FTL technique cannot fully reduce write activities compared to the baseline scheme *h*FTL, especially after garbage collection happens. The reason is analyzed and we concluded that WAP-FTL does not consider the behavior of I/O requests, sequential or random, may introduce extra valid page copy overhead in garbage collection. These observations motivate us to further extend this work for reducing write activities in PCM-based embedded systems.

CHAPTER 4

PCM-FTL: A TWO-LEVEL PCM-AWARED FLASH MEMORY MANAGEMENT TECHNIQUE

4.1 Overview

Phase change memory (PCM) is considered as DRAM replacement for designing main memory in embedded systems [21, 26, 52, 77, 78, 101]. However, compared to DRAM, PCM can only endure 10^6 to 10^8 writes per cell [38]. As main memory is one of the most heavily accessed components in embedded systems, the limited endurance of PCM leads to a shortened memory lifetime especially for write-intensive requests. It is therefore necessary to eliminate redundant write activities in PCM-based embedded systems. On the other hand, with the advantages of small size, shock resistance, and low power, NAND flash memory is widely used as a secondary storage and has been integrated into PCM-based embedded systems [48, 68, 87]. As a result, how to effectively manage NAND flash memory and avoid a fast worn-out of PCM-based embedded systems should be taken into account. Therefore, this chapter focuses on exploring the management of NAND flash memory in a PCM-based embedded system, while considering write activities in PCM to increase the reliability of the system.

In Chapter 3, we proposed a write-activity-aware page-level flash management technique, WAP-FTL, for reducing write activities in PCM-based embedded systems. However, the experimental results show that our WAP-FTL technique cannot fully reduce write activities compared to the baseline scheme, especially after garbage collection happens. The reason is that WAP-FTL does not consider the access behavior of I/O requests, which mainly consists of sequential and random requests. In WAP-FTL, extra page copy is introduced in

garbage collection and causes significant bit flips in FTL mapping table. Therefore, these observations motivate us to further extend the proposed write-activity-aware strategy in WAP-FTL for effectively reducing write activities in PCM-based embedded systems.

In this chapter, we propose a write-activity-aware two-level flash memory management technique, named **PCM-FTL**, to effectively manage NAND flash memory and enhance the endurance of PCM-based embedded systems meanwhile, with the advantage that no changes are required to the file system, or hardware implementation of the NAND/PCM chip. Our basic idea is to preserve each bit in FTL mapping table, which is stored in PCM, from being inverted frequently, i.e., we focus on reducing the number of bit flips in a PCM cell when updating the FTL mapping table. Unlike WAP-FTL proposed in Chapter 3, PCM-FTL employs a two-level mapping mechanism, which not only focuses on reducing write activities of PCM but also considers the access behavior of I/O requests.

To achieve this, in PCM-FTL, 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. Similarly, by utilizing the fine-grained hardware optimization technique for eliminating a write if its designated PCM cell holds the same value [52, 101], PCM-FTL actively chooses a physical block in NAND flash memory whose physical block number incurs minimum number of bit flips in PCM cells so as to write the different bit into PCM. Consequently, the write activities are reduced and the lifetime of PCM-based embedded systems is enhanced. To the best of our knowledge, PCM-FTL is the first effective technique proposed for managing NAND flash memory in PCM-based embedded systems with the consideration of write activities.

Similar as WAP-FTL, we conduct experiments with the same set of realistic I/O traces collected from notebook and Google Android^{TM}platform. A representative FTL design *h*FTL [48] for PCM-based embedded systems is selected as a baseline scheme. The proposed PCM-FTL is compared with *h*FTL in terms of the maximum and total number of bit flips in PCM cells with various configurations. For the DiskMon traces collected from notebook, the experimental results show that PCM-FTL reduces the maximum number of bit

flips among PCM cells by 93.10% (83.10%) on average, and reduces the total number of bit flips of all PCM cells by 64.00% (70.90%) on average in a PCM-based embedded system with 1GB (4GB) NAND flash memory. For the Google Android^{TM}traces, the experimental results show that PCM-FTL reduces the maximum number of bit flips among PCM cells by 99.82% (99.86%) on average, and reduces the total number of bit flips of all PCM cells by 93.10% (98.17%) on average in a PCM-based embedded system with 1GB (4GB) NAND flash memory.

This chapter makes the following contributions:

- We present for the first time a write-activity-aware two-level flash memory management technique to effectively manage NAND flash memory and enhance the lifetime of PCM-based embedded systems by reducing redundant write activities.
- We demonstrate the effectiveness of our technique by comparing with a representative FTL using a set of realistic I/O workloads.

The rest of this chapter is organized as follows. Section 4.2 discusses the background of system architecture and motivation. Section 4.3 presents our proposed PCM-FTL technique. Section 4.4 presents the experimental results. Finally, we conclude the chapter in Section 4.5

4.2 Motivation and Background

In this section, we first introduce the motivation of our work. Then we present the background knowledge of PCM-based embedded systems studied by this chapter.

4.2.1 Motivation

As depicted in Chapter 3, *h*FTL [48], a page-level mapping FTL scheme designed for the PCM-based embedded systems, does not consider the write activities in PCM and impose

lifetime issue for the PCM-based embedded systems. Though WAP-FTL with a write-activity-aware strategy is proposed to minimize bit flips during the update process of page-level mapping table each time, it does not consider the access behavior of I/O requests and also induce extra valid page copy overhead when garbage collection happens. The experimental results reported that WAP-FTL cannot fully reduce write activities for all realistic I/O traces. Since PCM cell can only sustain limited number of write cycles, frequent update operations in mapping table will lead to the fast worn out of PCM. Therefore, these observations motivate us to propose a new technique that can effectively reduce write activities and enhance the lifetime of PCM-based embedded systems as well.

4.2.2 PCM-Based Embedded Systems

In this chapter, we target at the PCM-based embedded system with the proposed write-activity-aware two-level flash memory management technique. As shown in Figure 4.1, a two-level mapping table with a block-level mapping table buffer and a page-level mapping table is maintained by the PCM-based main memory. For reducing write activities of the mapping table in PCM, the proposed PCM-FTL scheme is integrated into the PCM-based embedded system to replace the original flash translation layer. For the coming read requests, PCM-FTL checks the two-level mapping table in PCM and obtain the corresponding physical page in NAND flash memory for reading. For the coming write requests, PCM-FTL serves the requests by allocating free pages in NAND flash memory and updates the corresponding mapping records of the physical pages in the two-level FTL mapping table in PCM.

4.3 PCM-FTL: PCM-Awared Two-Level FTL

In this section, we present the details of our PCM-FTL, a write-activity-aware two-level flash memory management technique, that can effectively enhance the endurance of the PCM-based embedded systems. We first present an overview of PCM-FTL in Section 4.3.1. We

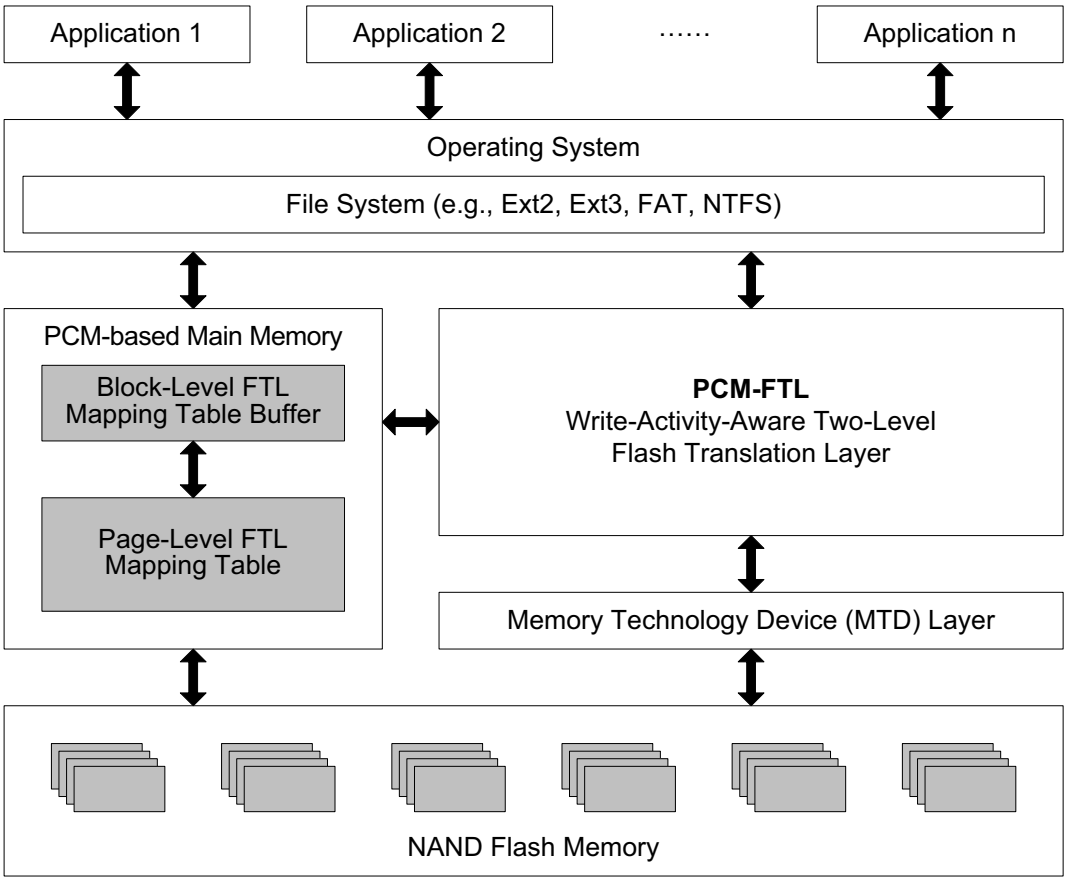


Figure 4.1. PCM-based embedded system with the proposed write-activity-aware two-level PCM-FTL technique.

then provide a detailed description of PCM-FTL in Section 4.3.2. Finally, a wear leveling method of PCM-FTL is presented in Section 4.3.3.

4.3.1 Overview of PCM-FTL

The objective of PCM-FTL is to reduce write activities in PCM-based embedded systems, and therefore, the endurance of PCM is enhanced. So the basic idea of PCM-FTL is to preserve each bit in FTL mapping table, which is stored in PCM, from being inverted frequently, i.e., we focus on reducing the maximum number of bit flips in a PCM cell when updating the FTL mapping table in PCM. Different from the previous work [48], our PCM-FTL adopts a two-level mapping mechanism, which not only focuses on minimizing write activities in

PCM but also considers the access behavior of I/O requests. PCM-FTL uses a page-level mapping table to record the mapping of random write requests not frequently updated, and allocates a tiny buffer of block-level mapping table to cache the mapping records of those most frequently updated sequential write requests. With the consideration of write activities, once a block is needed for incoming write requests, PCM-FTL actively chooses a physical block in NAND flash memory whose physical block number incurs minimum number of bit flips in PCM cells. By applying PCM-FTL, the number of bit flips is reduced, and thus the number of write activities in PCM is minimized. Consequently, the endurance of the PCM-based embedded system is enhanced.

4.3.2 PCM-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 sequential write requests. Otherwise, without considering the access behavior of I/O workload, we can not effectively manage NAND flash memory and may waste lots of blocks in garbage collection due to frequent update operations. Therefore, in PCM-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, 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 4.2 shows the structure of the proposed PCM-FTL. As shown, PCM-FTL first separates the I/O workload into random requests and sequential requests in terms of the predefined threshold. Then PCM-FTL adopts a two-level FTL mapping mechanism to handle these two cases as follows:

- For random requests: PCM-FTL consecutively allocates physical pages from the first

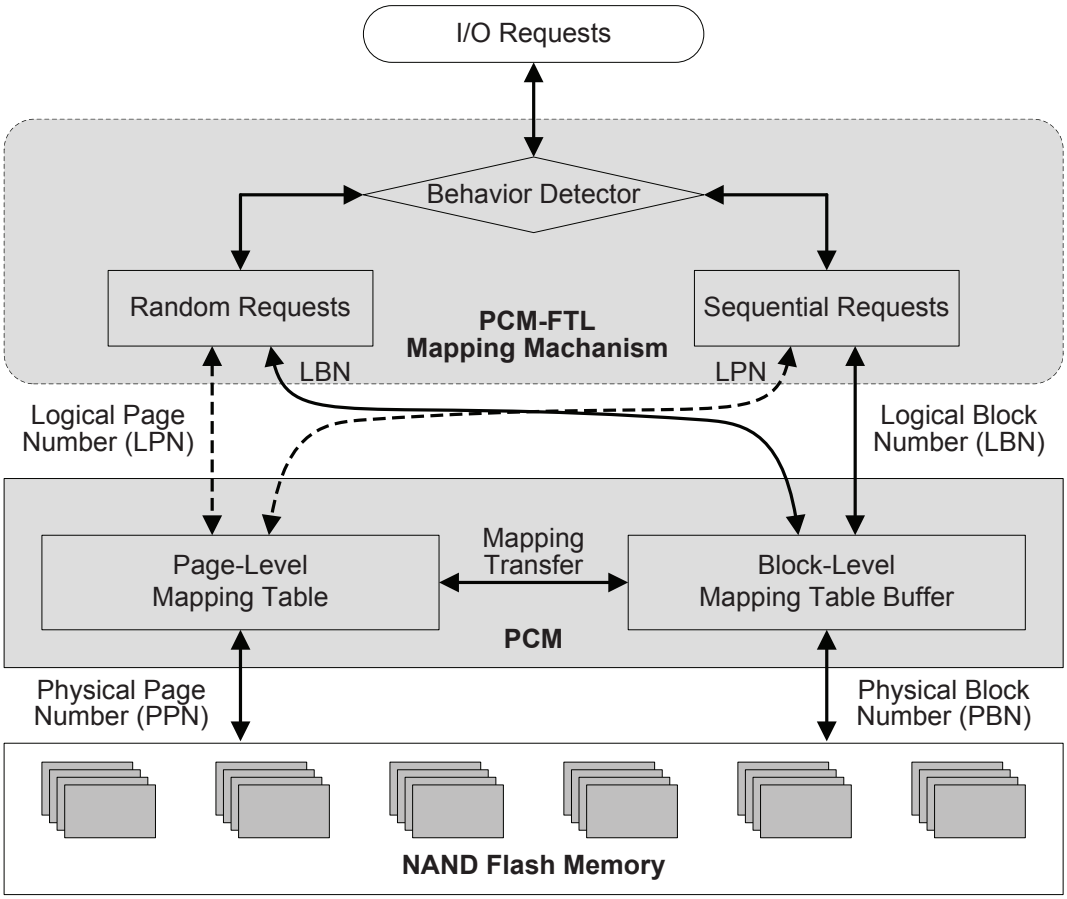


Figure 4.2. Illustration of PCM-FTL write-activity-aware two-level mapping mechanism.

page of a physical block in NAND flash memory, so that all pages in blocks are fully utilized. Accordingly, PCM-FTL adds LPN to PPN mapping record of random requests into the page-level mapping table in PCM.

- For sequential requests: PCM-FTL allocates physical pages of a physical block 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, PCM-FTL adds an LBN to PBN mapping record of sequential requests into the block-level mapping table buffer in PCM.

In PCM-FTL, we only allocate a tiny buffer for tentatively storing a part of the block-level mapping table. For example, the size of this block-level mapping table buffer can be set

as 5%, 10% or 20% of the size of the original block-level mapping table, in order to achieve an acceptable space overhead. 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 record of LBN to PBN is first expanded to the corresponding LPNs to PPNs mapping records, and then these page-level mapping records are put back 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, so N_p entries in page-level mapping table should be filled with the corresponding N_P LPN to PPN mapping records 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 the block-level mapping table buffer which only contributes a small number of bit flips in PCM. The experimental results in Section 4.4 confirm this fact.

To further reduce write activities in PCM-based embedded systems, we adopt the similar write-activity-aware strategy used in WAP-FTL. In our PCM-FTL, 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 logical page number (LBN). Then according to the original PBN, we actively select a physical block in NAND flash memory whose PBN has the minimum number of Hamming distance with the original PBN, i.e., we try to find a new PBN to achieve a 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 can be reduced, and the endurance of PCM is enhanced.

Algorithm 4.3.1 shows the process of a write operation of PCM-FTL. PCM-FTL first divides the incoming I/O requests into random writes or/and sequential writes according to a predefined threshold. Then the random and sequential write requests are processed

Algorithm 4.3.1 The algorithm of PCM-FTL

Input: I/O requests with random request or/and sequential request.
Output: Allocate pages for the I/O request.

Divide the I/O request into random writes or/and sequential writes according to a predefined threshold.
if Random write request arrives **then**
    Obtain the LBN and LPN of the random write request.
    if LBN's mapping is not in block-level mapping table buffer or LPN's mapping is not in page-level mapping table **then**
        This is a new write, allocate a new block PBN, and write the contents into the block sequentially from the first page.
        Add the mapping of (LPN, PPN) into the page-level mapping table.
    end if
    if LBN's mapping exists in block-level mapping table buffer or LPN's mapping exists in page-level mapping table **then**
        This is an update, obtain the PBN of the updated block.
        if There exists enough space in the PBN block for the update request **then**
            Write the update contents in the left space of the PBN block sequentially, and invalid the old pages in the same block.
        else
            Actively find a new block whose block number is almost the same as PBN, write the update contents in the new block sequentially, and invalid the old pages in PBN block.
        end if
        Update block-level mapping table buffer or page-level mapping table.
    end if
end if
if Sequential write request arrives **then**
    Obtain the LBN and LPN of the sequential write request.
    if LBN's mapping is not in block-level mapping table buffer or LPN's mapping is not in page-level mapping table **then**
        This is a new write, allocate a new block PBN, and write the contents of the request into the block based on block offset.
        if The block-level mapping table buffer is full **then**
            Kick out least frequently used entry, add the kicked out mappings into page-level mapping table.
        end if
        Add the mapping of (LBN, PBN) into the block-level mapping table buffer.
    end if
    if LBN's mapping exists in block-level mapping table buffer or LPN's mapping exists in page-level mapping table **then**
        This is an update, obtain the PBN of the updated block.
        if There exists enough space in the PBN block for the update request **then**
            Write the update contents in the left space of the PBN block based on block offset, and invalid the old pages in the same block.
        else
            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 invalid the old pages in PBN block.
        end if
        Update block-level mapping table buffer or page-level mapping table.
    end if
end if

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 PCM-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, PCM-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.

An example of PCM-FTL is shown in Figure 4.3. This example is based on the I/O requests and the NAND flash memory assumptions for the motivational example shown in Figure 3.2. As shown, for the first random request with LPN (#18), we find a new block (PBN #00), and the content A is written consecutively into the first page (#00000) of block (PBN #00). For this request, there is no bit flip when updating the mapping table as we assume the mapping table is empty at the beginning. 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 PCM-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 consecutively into this block.

It is noticed that the 5th to 12th requests form a sequential write in a block, then we allocate a new block (PBN #11) for this request, and write the contents into each page of the block based on block offset. The corresponding LBN to PBN mapping (01, 11) is added

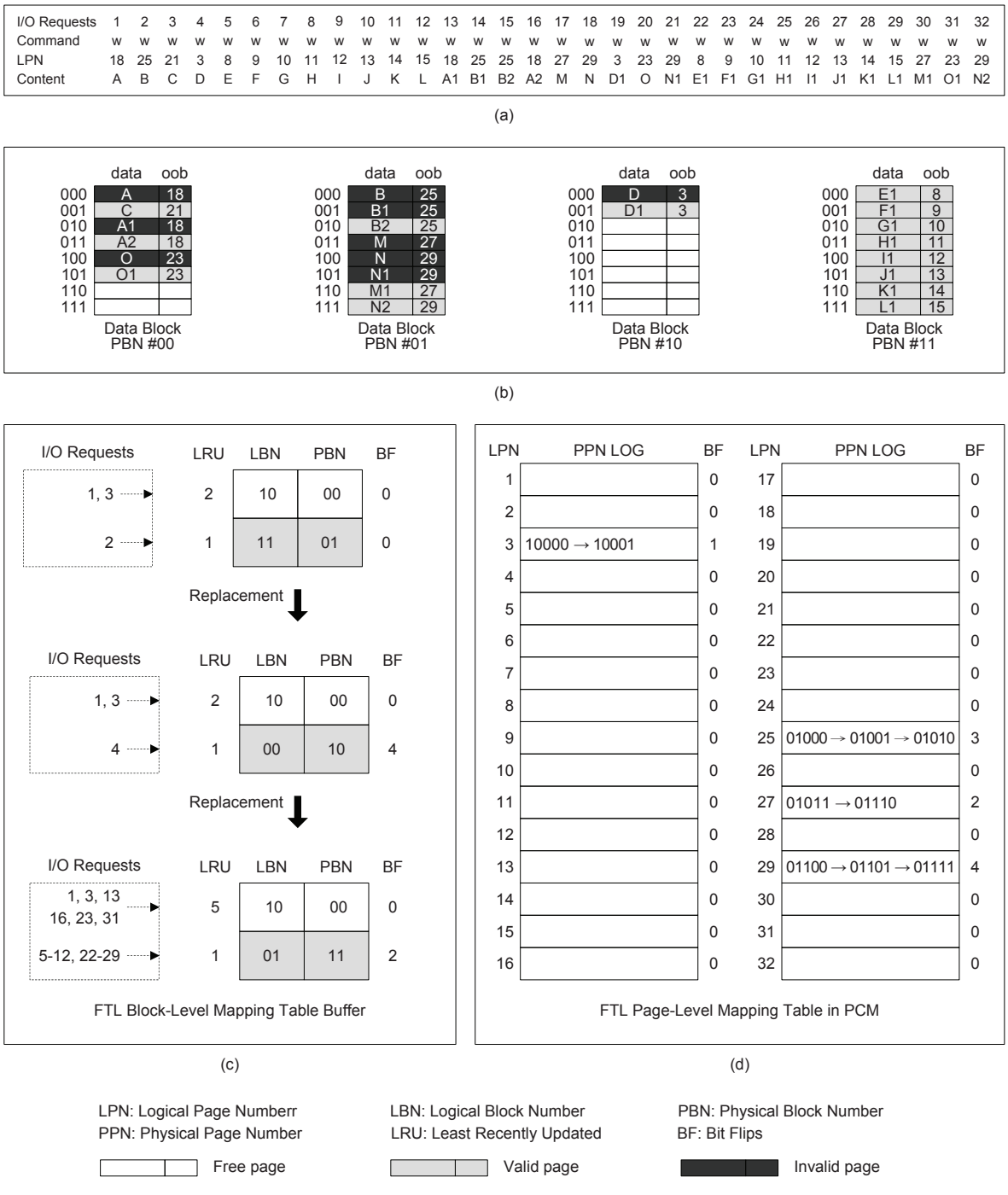


Figure 4.3. Illustration of PCM-FTL. (a) The status variation of blocks in NAND flash memory according to the access sequence in Figure 3.2. (b) The status variation of FTL page-level mapping table and block-level mapping table buffer in PCM.

into the block-level mapping table buffer. Later, when the following 22th to 29th sequential update requests arrive, then the old pages in the block (PBN #11) are invalidated. Since we cannot find free block, the block (PBN #11) is erased for reclaiming free pages, and the new update data E1 to L1 is written into this block based on block 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 find that the total number of bit flips in PCM is 16 by our PCM-FTL, while the total number of bit flips in PCM is 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 PCM. The experimental results in Section 4.4 also show that our scheme can effectively reduce the total number of bit flips in PCM cells.

4.3.3 PCM-FTL Wear Leveling Scheme

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 among all PCM cells. To avoid this scenario and enhance PCM endurance, a wear leveling method is integrated into PCM-FTL. Figure 4.4 demonstrates the process of our wear leveling method. As shown Figure 4.4(a), the block-level mapping table buffer becomes hot after buffering the frequently updated sequential write requests. However, the page-level mapping table is cold as it only serves the infrequently updated random write requests. Therefore, as shown in Figure 4.4(b), during a period of time (e.g., every 2000 I/O requests), the block-level mapping table buffer is moved across the whole page-level mapping table region in PCM chip. Since the migration of the block-level mapping table buffer is infrequent, the number of copy operations of mapping records is acceptable. Finally, in Figure 4.4(b), we can see that the even distribution of write activities (i.e., bit flips) across the whole two-level mapping table region in PCM is obtained. The detailed description of our wear leveling scheme is shown in Algorithm 4.3.2. The experimental results in Section 4.4 confirm the effectiveness of our scheme, and also show that our PCM-FTL can achieve better wear leveling over different traces.

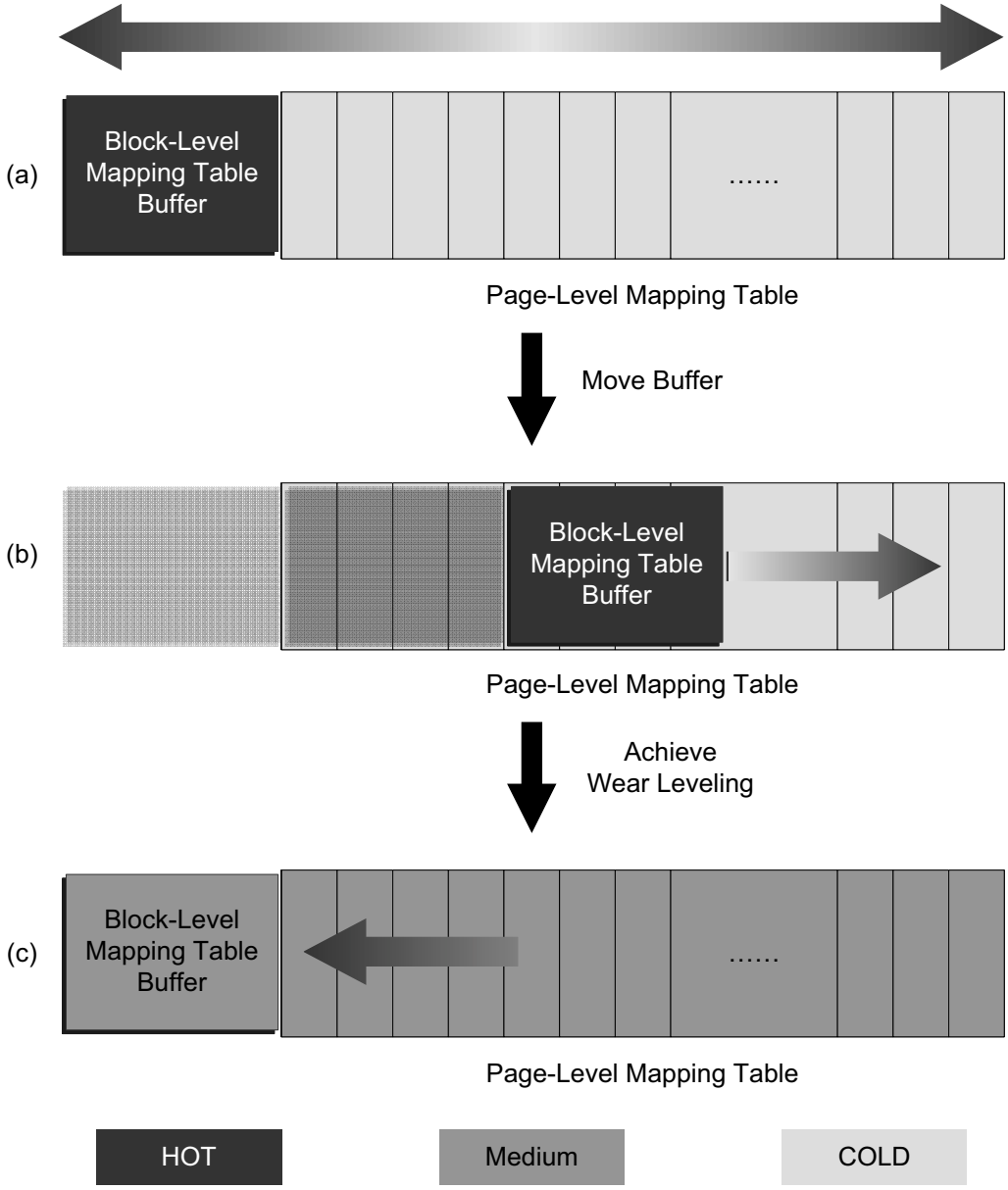


Figure 4.4. Illustration of the wear leveling method adopted by PCM-FTL. (a) The initial status of the two-level mapping table with uneven distribution of write activities. (b) Move block-level mapping table buffer across the whole page-level mapping table to achieve wear leveling. (c) Write activities are evenly distribute among the two-level mapping table after moving block-level mapping table buffer.

Algorithm 4.3.2 The algorithm of wear leveling in PCM-FTL

Input: The number of writes conducted in PCM cells so far, the allowed number of writes in PCM cells before triggering wear leveling operations.

Output: Next position of the first PCM entry of block buffer.

```
1: WEAR_LEVELING_THRESHOLD ← The allowed number of writes in PCM cells before triggering wear
   leveling operations.
2: WL_Counter ← The number of writes conducted in PCM cells so far.
3: Current_Offset ← Current position of the first PCM entry of block buffer.
4: Next_offset ← Next position of the first PCM entry of block buffer.
5: PTE ← PCM entry of page-level mapping table.
6: BTE ← PCM entry of block buffer.
7: if WL_Counter < WEAR_LEVELING_THRESHOLD then
8:     WL_Counter++.
9:     RETURN.
10: else
11:    Next_Offset = (Current_Offset + block buffer length)%(Total PCM length - block buffer length).
12:    PTE ← First PCM entry of the page-level mapping table starting from Next_Offset.
13:    BTE ← First PCM entry of block buffer starting from Current_Offset.
14:    for each PCM entry in block buffer do
15:        Exchange the content of PTE and BTE.
16:        PTE ← Next PCM entry of the page-level mapping table.
17:        BTE ← Next PCM entry of the block buffer.
18:    end for
19:    Reset WL_Counter to 0.
20:    RETURN Next_Offset.
21: end if
```

4.4 Evaluation

To evaluate the effectiveness of the proposed PCM-FTL, we conduct a series of experiments and present the experimental results with analysis in this section. We compare and evaluate our proposed PCM-FTL scheme over the representative page-level FTL scheme, *h*FTL [48],

based on the maximum and total number of bit flips in PCM cells. We also compare the wear leveling results of PCM-FTL and *h*FTL.

In this chapter, we assume that the FTL mapping table is stored in a single-level cell (SLC) PCM (i.e., a PCM cell holds only one bit), and the user data is stored in a multi-level cell (MLC) NAND flash memory, which is widely used in embedded systems.

4.4.1 Experimental Setup

In this chapter, we adopt the same experimental setup configuration as that given by Table 3.1 in Chapter 3. The evaluation is also conducted through a trace-driven simulation framework, in which a simulator is designed to evaluate PCM-FTL and *h*FTL using a variety of realistic I/O traces collected from notebook and Google Android platform, respectively. These realistic I/O traces, i.e., CopyFiles, DownFiles, Office, P2P, Communications, Internet, MixedApps, and Multimedia, are introduced in detail in Chapter 3. We use these eight I/O traces to evaluate our PCM-FTL below.

The framework of our simulation platform is shown in Figure 3.6. This simulation framework simulates our PCM-FTL management scheme over the PCM-based embedded systems, which consists of a NAND flash memory and a PCM for storing our two-level mapping table. In our experiments, the traces along with various parameters of NAND flash memory, such as block size, page size, etc, are fed into our simulator. The page size, number of pages in a block, and size of the OOB for each page are set as 2KB, 64, and 64 Bytes, respectively. To fully evaluate our technique, we conduct the experiments on a PCM-based embedded system with 1GB and 4GB NAND flash memory, respectively.

4.4.2 Results and Discussion

In this section, we present the experimental results of the proposed PCM-FTL and the baseline scheme with analysis. We first present the experimental results of PCM-FTL with various parameter configurations. Then we present the endurance comparison of PCM-FTL and

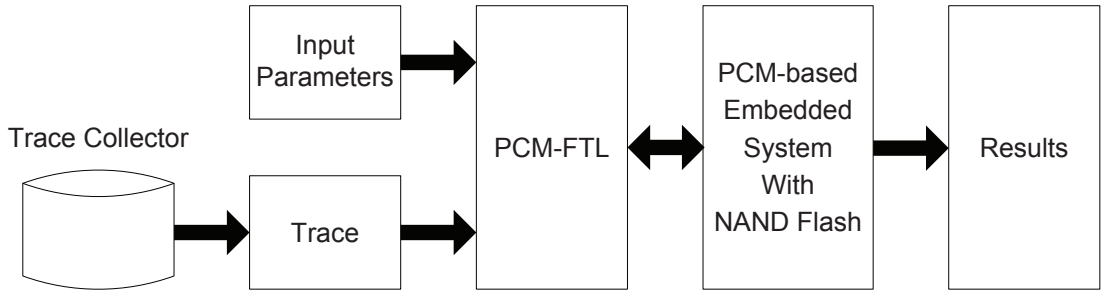


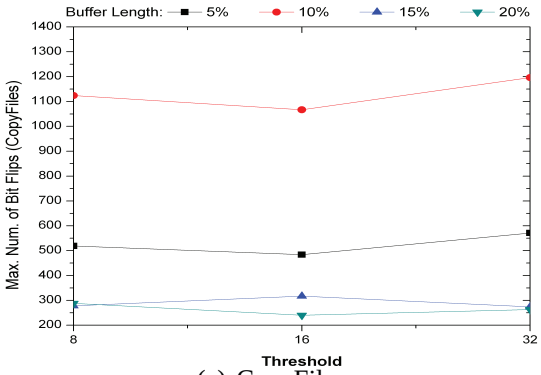
Figure 4.5. The framework of simulation platform for evaluating the proposed PCM-FTL technique.

*h*FTL. Finally, we present the wear leveling comparison of PCM-FTL and *h*FTL. Experiments are conducted based on our PCM-based embedded system simulator with 1GB and 4GB NAND flash memory over eight distinct traces gathered from notebook and Google Android^{TM}platform.

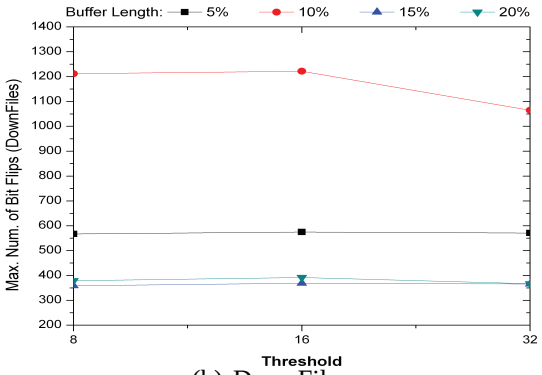
Impact of Threshold and Buffer Size

In PCM-FTL, there are mainly two parameters predefined for achieving write activity reduction with limited overhead. These two parameters are the threshold and the buffer size. The threshold is used for determining whether a request is random or sequential by comparing the request length with the threshold value, and the buffer size is used for determining the size of the block-level mapping table buffer.

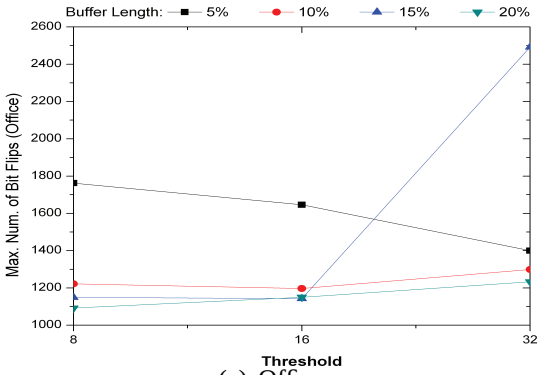
In order to show how the maximum and total number of bit flips in PCM cells are influenced by different combinations of threshold value and buffer size, and to find a suitable combination of these two parameters for the following evaluations, we first conduct experiments by varying the above parameters with different values. The candidate threshold values are set as 8, 16 and 32, while the candidate buffer size is set as 5%, 10%, 15% and 20% of the original size of block-level mapping table. Physically, in PCM-based main memory, the block-level mapping table buffer follows the page-level mapping table. For each threshold value, we run experiments over the eight traces by combining one of the three candidates of buffer size. We collect the maximum number of bit flips in each PCM cell and total number



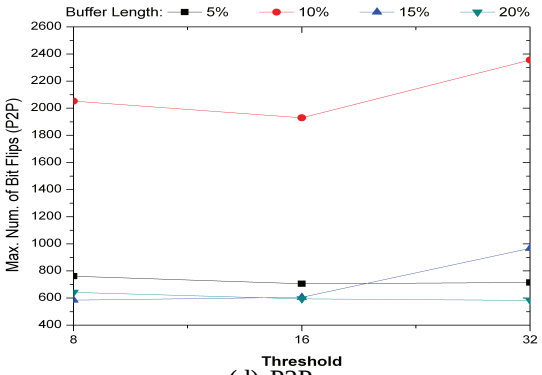
(a) CopyFiles



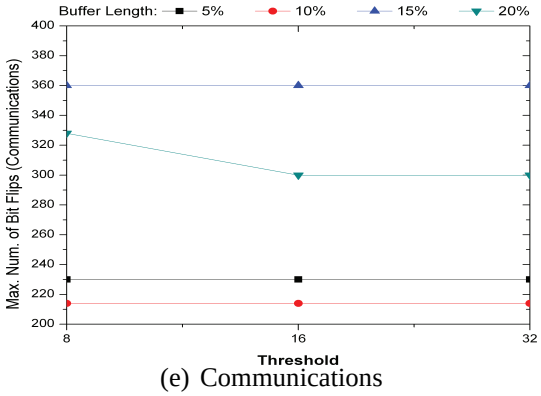
(b) DownFiles



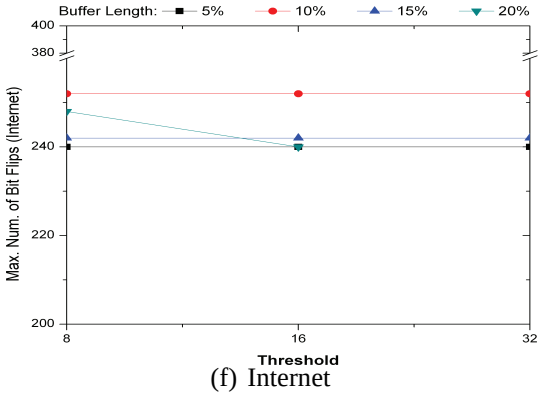
(c) Office



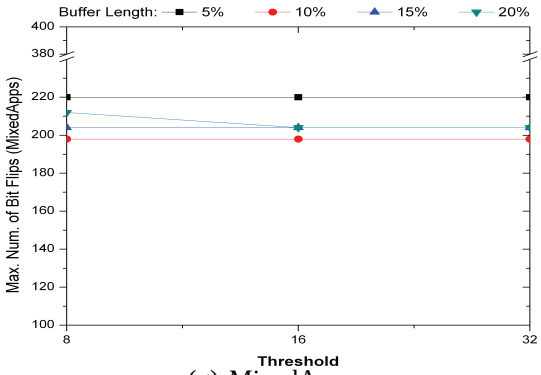
(d) P2P



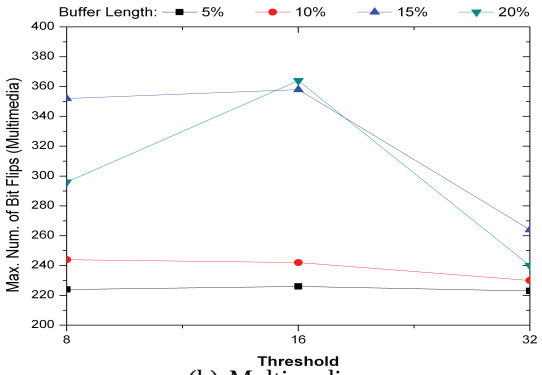
(e) Communications



(f) Internet

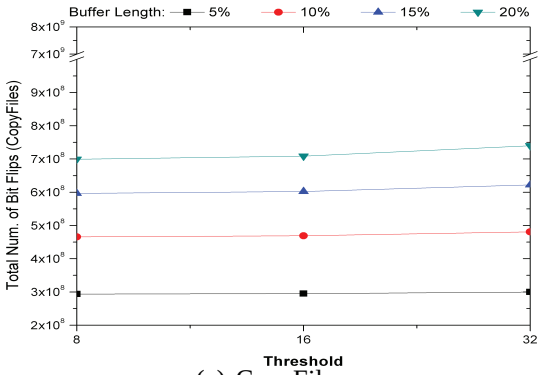


(g) MixedApps

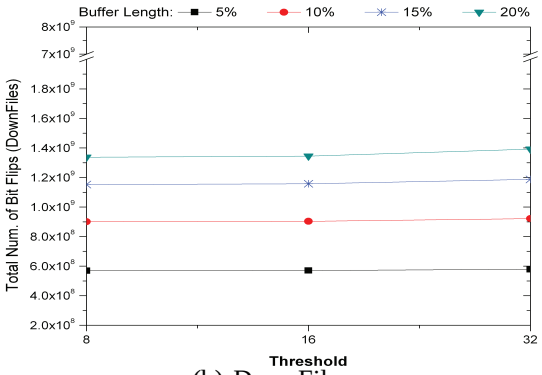


(h) Multimedia

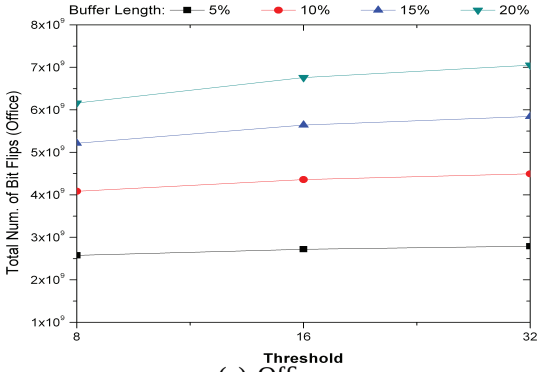
Figure 4.6. The maximum number of bit flips obtained from the PCM-based embedded systems with 1GB NAND flash memory for PCM-FTL with different parameter combinations.



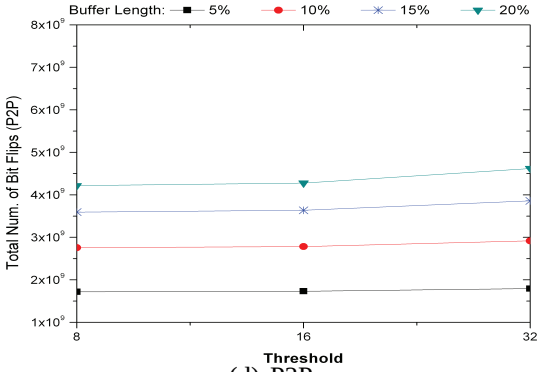
(a) CopyFiles



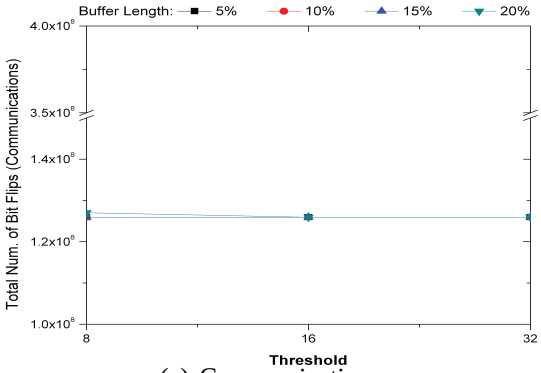
(b) DownFiles



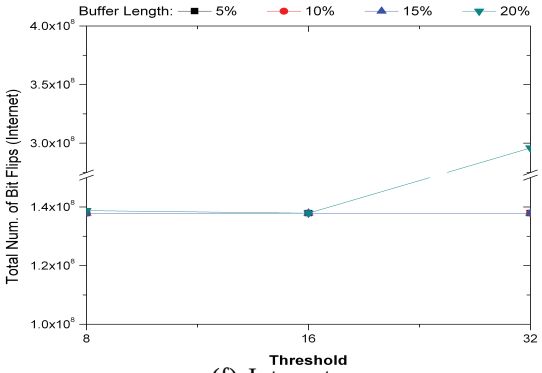
(c) Office



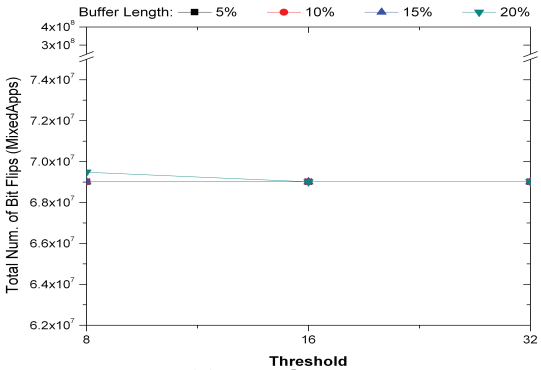
(d) P2P



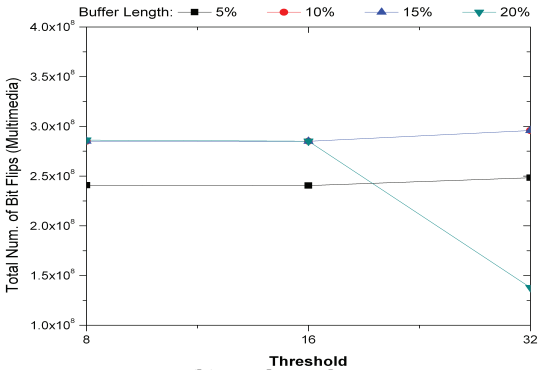
(e) Communications



(f) Internet



(g) MixedApps



(h) Multimedia

Figure 4.7. The total number of bit flips obtained from the PCM-based embedded systems with 4GB NAND flash memory for PCM-FTL with different parameter combinations.

of bit flips across the whole mapping table region in PCM chip. The buffer size is triggered to move across the whole page-level mapping table region in the PCM chip for every 2800 writes to PCM cells.

The results are shown in Figure 4.6 and Figure 4.7 for PCM-FTL with different combinations of threshold values and buffer sizes over eight realistic I/O traces in the PCM-based embedded systems with 1GB NAND flash memory. In Figure 4.6 and Figure 4.7, we can see that for each trace, the similar trends are obtained for both results of the maximum and total number of bit flips. Although the plots for different traces are different, most of them show the same trend: the maximum and total number of bit flips is not changed a lot when the threshold varies in a range (from 8 to 32), and then shows slightly increase with the increase of the threshold value, except for Figure 4.6 (c), Figure 4.6(h), and Figure 4.7(h). The difference of these three sub-figures maybe caused by the access pattern of the corresponding trace. In Figure 4.6, for all traces, we can see that a smaller number of the maximum number of bit flips can be achieved when the threshold value is set as 8 and the buffer size is set as 5%. In Figure 4.7, for all traces, we can also find that a smaller number of the total number of bit flips can be achieved if the threshold value is set as 8 and the buffer size is set as 5%. Since the buffer size is only 5% of the original size of the block-level mapping table, which is also very small compared to page-level mapping table, so the space overhead of our PCM-FTL is negligible. Therefore, we set the threshold value as 8 and the buffer size as 5% in the following experiments. The experimental results obtained from the PCM-based embedded systems with 4GB NAND flash memory show the similar trend, so we ignore the results here.

PCM Endurance

The objective of this chapter is to reduce write activities to enhance the lifetime of PCM-based embedded systems. Therefore, the lifetime of PCM is one of the most important factors in analyzing the reliability of PCM-based embedded systems. As mentioned before, PCM lifetime is mainly influenced by the maximum number of bit flips in a PCM cell,

Table 4.1. PCM-FTL versus *h*FTL in terms of the total and maximum number of bit flips in PCM cells. (1GB NAND flash memory, threshold = 8, buffer size = 5%)

Trace Name	Total Num. of Bit Flips			Maximum Num. of Bit Flips		
	*h*FTL	PCM-FTL	PCM-FTL over *h*FTL	*h*FTL	PCM-FTL	PCM-FTL over *h*FTL
CopyFiles	559496658	293866292	47.48%	9977	519	94.80%
DownFiles	1756464372	568987257	67.61%	21945	567	97.42%
Office	7520028995	2576892175	65.73%	9385	1762	81.23%
P2P	6929967624	1718812456	75.20%	74540	762	98.98%
Average			64.00%			93.10%
Communications	9883730134	125943263	98.73%	158029	230	99.85%
Internet	9555187107	137866567	98.56%	134281	240	99.82%
MixedApps	6591084832	69012523	98.95%	139241	220	99.84%
Multimedia	8347917692	240771631	97.12%	95621	224	99.77%
Average			98.34%			99.82%

i.e., the maximum number of bit flips in a PCM cell determines the lifetime of PCM. 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 reducing write activities (total number of bit flips) in PCM cells but also reducing the maximum number of bit flips for each PCM cell. Table 4.1 and Table 4.2 report the experimental results of PCM-FTL and the baseline scheme *h*FTL over eight realistic I/O traces in terms of the maximum and total number of bit flips among all PCM cells in PCM-based embedded systems with 1GB NAND flash memory and 4GB NAND flash memory, respectively.

We observe that PCM-FTL can significantly reduce write activities in PCM cells in comparison with the baseline scheme *h*FTL. As shown in Table 4.1, compared to *h*FTL over

Table 4.2. PCM-FTL versus *h*FTL in terms of the total and maximum number of bit flips in PCM cells. (4GB NAND flash memory, threshold = 8, buffer size = 5%)

Trace Name	Total Num. of Bit Flips			Maximum Num. of Bit Flips		
	*h*FTL	PCM-FTL	PCM-FTL over *h*FTL	*h*FTL	PCM-FTL	PCM-FTL over *h*FTL
CopyFiles	122605530	93456325	23.77%	10461	2076	80.15%
DownFiles	842401436	191579924	77.26%	20857	3923	81.19%
Office	6981790260	919567590	86.83%	36667	7377	79.88%
P2P	15269865958	650611634	95.74%	86383	7623	91.18%
Average			70.90%			83.10%
Communications	10305765684	127165676	98.77%	158029	182	99.88%
Internet	9988536600	138654167	98.61%	134281	176	99.87%
MixedApps	5130339159	69580586	98.64%	149439	174	99.88%
Multimedia	8654772875	288691259	96.66%	95846	186	99.81%
Average			98.17%			99.86%

the first four DiskMon traces, PCM-FTL can achieve an average reduction of 93.10% and a maximum reduction of 98.98% in the maximum number of bit flips in the PCM-based embedded system with 1GB NAND flash memory. Moreover, PCM-FTL can achieve an average reduction of 64% and a maximum reduction of 75.2% in terms of the total number of bit flips. For the Google Android™traces, we can see that PCM-FTL can achieve more reduction of the maximum and total number of bit flips. An average reduction of 99.82% (98.34%) and a maximum reduction of 99.85% (98.95%) in the maximum (total) number of bit flips are obtained.

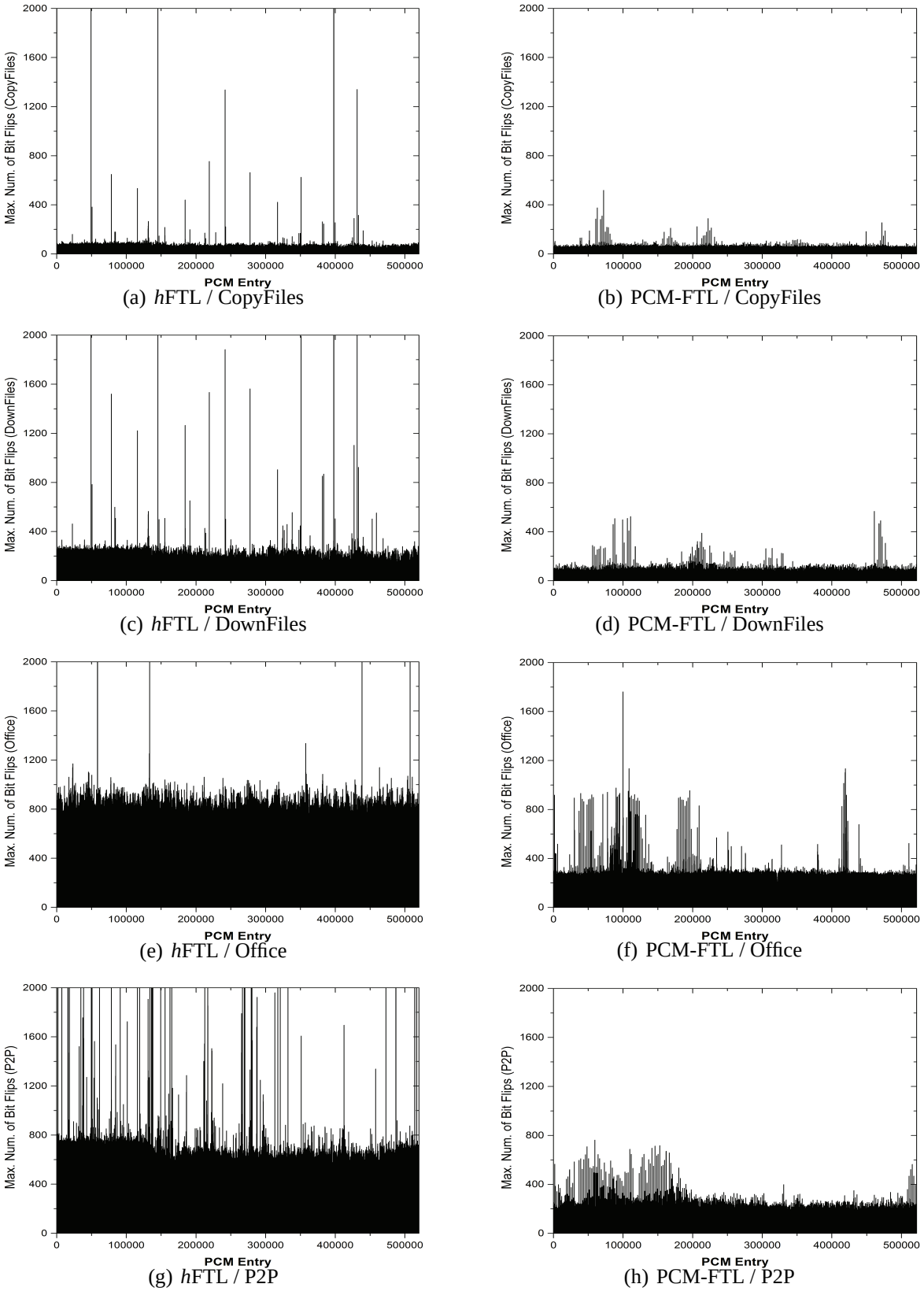
Table 4.2 shows the similar results for the comparison of PCM-FTL and *h*FTL in the PCM-based embedded system with 4GB NAND flash memory. For the first four DiskMon

traces, PCM-FTL can achieve an average reduction of 83.10% and a maximum reduction of 91.18% in the maximum number of bit flips. In terms of the total number of bit flips, PCM-FTL can achieve an average reduction of 70.90% and a maximum reduction of 95.74%. For the Google Android™traces, we can see that PCM-FTL can an achieve average reduction of 99.86% (98.17%) and a maximum reduction of 99.88% (98.77%) in the maximum (total) number of bit flips. As shown, the above experimental results show that our PCM-FTL significantly reduces write activities in PCM cells. Therefore, in the PCM-based embedded systems, by applying PCM-FTL, the lifetime of PCM can be enhanced.

PCM Wear Leveling

Wear leveling is another one of the most important factors that influence the lifetime of the PCM-based embedded systems. Figure 4.8 and Figure 4.9 show the distribution of the maximum number of bit flips among all mapping table entries in a PCM-based embedded system with 1GB NAND flash memory over the DiskMon traces and Google Android™traces, respectively. For each sub-figure, the x-axis shows the total number of mapping entries inside the page-level mapping table and block-level mapping table buffer in PCM, and the y-axis shows the maximum number of bit flips extracted from each mapping entry of the page-level mapping table and the block-level mapping table buffer. To present the distributions clearly, we restrict the maximum number of bit flips on y-axis to 2,000.

As shown in Figure 4.8, for *h*FTL scheme, we observe that the distribution of the maximum number of bit flips varies a lot, and this may impose a fast worn-out of PCM cells. Compared with *h*FTL, by adopting our wear leveling method described in Section 4.3.2, PCM-FTL distributes the maximum number of bit flips more evenly among all PCM cells. Though the distribution of the maximum of bit flips obtained over the trace Office is not so even, it is still better than that of *h*FTL, and we can further obtain better distribution by tune the parameters of PCM-FTL. In Figure 4.9, we can see that the write activities of *h*FTL are mapped to a specific region in PCM due to access pattern of I/O requests, and the maximum number of bit flips is greater than 2,000. In contrast, PCM-FTL achieves an even



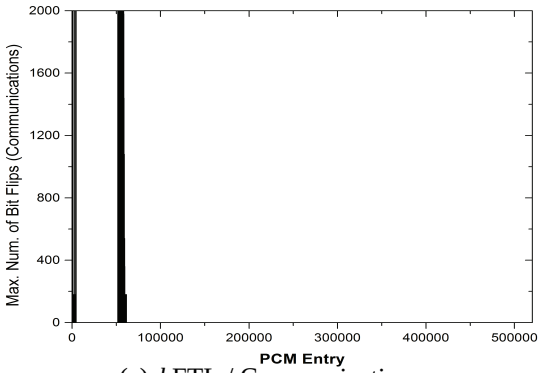
(a) *h*FTL / CopyFiles (b) PCM-FTL / CopyFiles

(c) *h*FTL / DownFiles (d) PCM-FTL / DownFiles

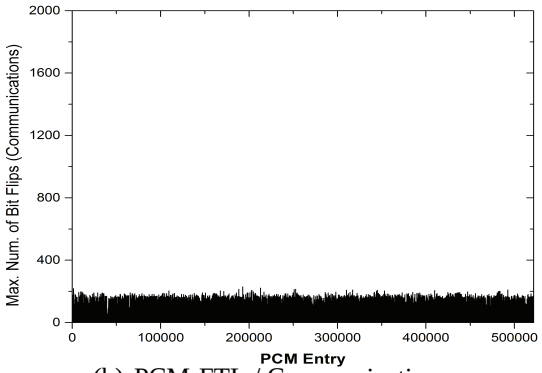
(e) *h*FTL / Office (f) PCM-FTL / Office

(g) *h*FTL / P2P (h) PCM-FTL / P2P

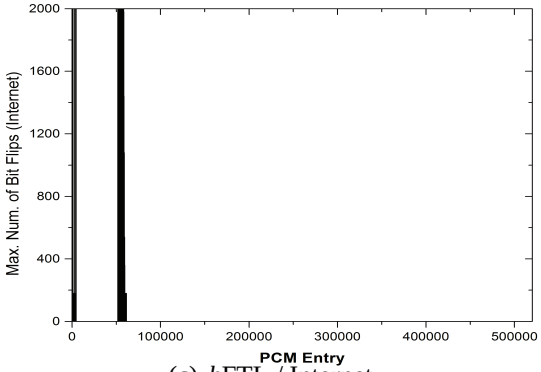
Figure 4.8. The wear leveling comparison of *h*FTL and PCM-FTL in a PCM-based embedded system with 1GB NAND flash memory over four realistic DiskMon traces.



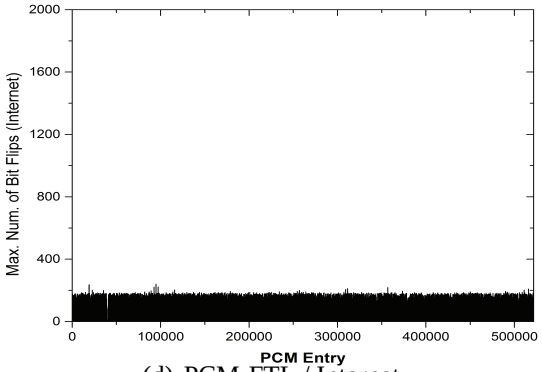
(a) *h*FTL / Communications



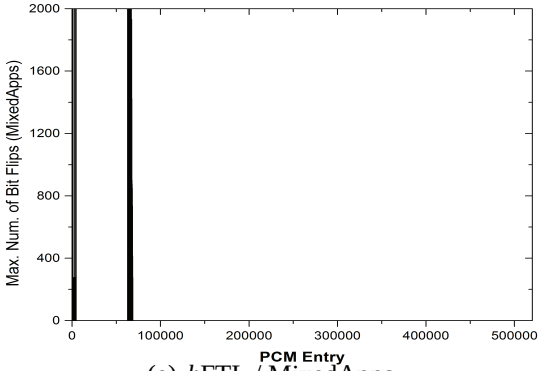
(b) PCM-FTL / Communications



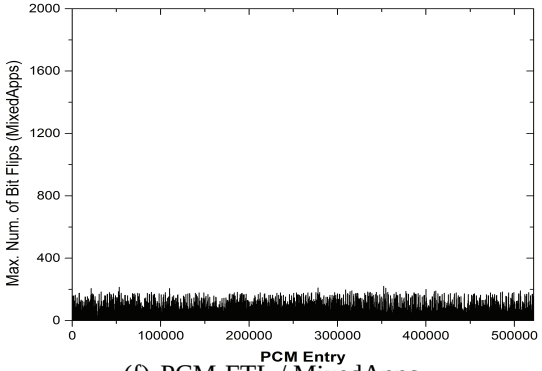
(c) *h*FTL / Internet



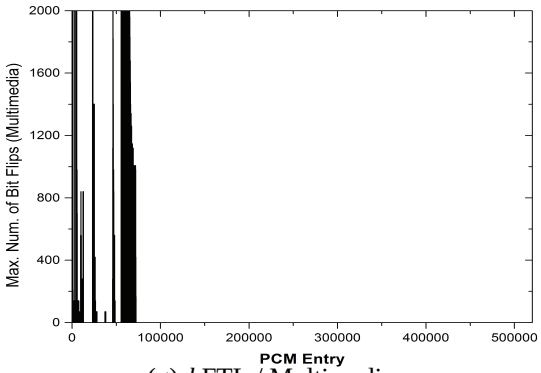
(d) PCM-FTL / Internet



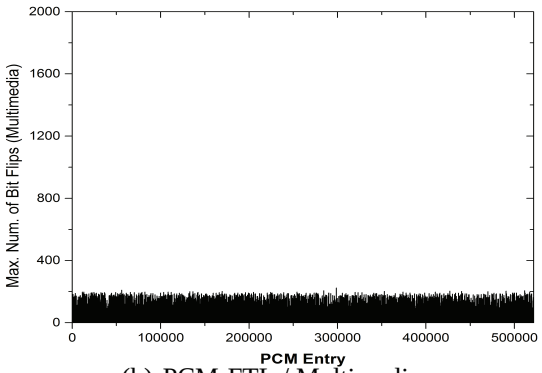
(e) *h*FTL / MixedApps



(f) PCM-FTL / MixedApps



(g) *h*FTL / Multimedia



(h) PCM-FTL / Multimedia

Figure 4.9. The wear leveling comparison of *h*FTL and PCM-FTL in a PCM-based embedded system with 1GB NAND flash memory over four realistic Google Android™ traces.

distribution of the maximum number of bit flips for the Google Android™traces, and almost all of them are below 400. In summary, PCM-FTL delivers dramatically better reliability than the baseline scheme.

4.5 Summary

In this chapter, we have proposed a write-activity-aware two-level flash memory management technique, named PCM-FTL, which takes the first step to significantly reduce write activities in PCM-based embedded systems with NAND flash memory. In our PCM-FTL, the performance improvement is achieved by preserving a bit in a PCM cell from being inverted frequently. Through a two-level mapping mechanism and a write-activity-aware strategy, unnecessary write activities in PCM are directly reduced. We conducted experiments on a set of realistic I/O workload collected by DiskMon and Google Android™. For the DiskMon traces, 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 in a PCM-based embedded system with 1GB (4GB) NAND flash memory. For the Google Android™traces, the experimental results show that the maximum number of bit flips among PCM cells can be reduced by 99.82% (99.86%) on average, and the total number of bit flips of all PCM cells can be reduced by 93.10% (98.17%) on average in a PCM-based embedded system with 1GB (4GB) NAND flash memory. Furthermore, the results show that PCM-FTL can evenly distribute write activities among all PCM cells in comparison with a representative baseline FTL scheme.

CHAPTER 5

WAB-FTL: A BLOCK-LEVEL PCM-AWARED FLASH MEMORY MANAGEMENT TECHNIQUE

5.1 Overview

As discussed in previous chapters, 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 [21, 26, 77, 96]. However, compared to DRAM, PCM can only sustain limited write operations (10^6 to 10^8 bit flips per cell) [38]. As main memory is a frequently accessed component, it is necessary to reduce redundant write activities in PCM to enhance the reliability of PCM-based embedded systems. On the other hand, with the advantages of small size, shock resistance, and low power, NAND flash memory is widely used as a secondary storage and has been integrated into PCM-based embedded systems [48, 68, 87]. How to avoid a fast worn-out of such emerging embedded systems and effectively manage NAND flash memory should be taken into account. Therefore, this chapter 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 use NAND flash memory, flash translation layer (FTL) is designed to emulate NAND flash memory as a disk drive, and logical addresses are mapped to physical addresses in NAND flash memory at a granularity of page-level or block-level [37, 51]. Following I/O requests, an FTL mapping table is employed to keep track of the continually updated mapping records. Many FTL schemes have been proposed [5, 6, 18, 29, 93], and most of them are mainly categorized into page-level scheme or block-level scheme according to the granularity of mapping unit [19]. 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 traditional 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. [48] propose a page-level FTL, namely *h*FTL, in which 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.

In Chapter 3 and Chapter 4, a write-activity-aware page-level and a write-activity-aware two-level flash memory management techniques have been proposed. However, page-level FTL scheme provides high performance but with significant memory requirement, so it may not be applicable for current PCM chips whose capacity is reported as 128Mb [61], 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 [6, 93]. For the example above, the memory requirement is only 0.3%. Though several block-level FTL schemes are proposed, none of them considers redundant write activities of block-level mapping table in PCM, either. 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 flash memory management technique to reduce write activities in PCM, such that the lifetime of the entire PCM-based embedded systems is enhanced.

In this chapter, we propose a **W**rite-**A**ctivity-aware **B**lock-level **FTL** design, 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, with the advantage that no changes are required to the file system, and hardware implementation of the NAND/PCM chip. Note that mapping records inside FTL mapping table are represented in a binary form in PCM. 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, we design a new merge strategy, called Lazy-Merge, to make our WAB-FTL scheme write activity aware. With Lazy-Merge strategy, the primary block is preserved from being erased in each merge operation. So the corresponding mapping record of the primary block in PCM remains unchanged. Its update is delayed until the corresponding primary block is erased in garbage collection for reclaiming more free blocks. On the other hand, the mapping record of the replacement block can be updated more frequently than that of the primary block, so in WAB-FTL, we further propose an additional tiny buffer, named Cooling-Pool, with multiple candidate mapping slots in PCM for caching the frequently updated mapping records to further reduce write activities in PCM. As mentioned in the previous chapters, several hardware optimization techniques for PCM have been developed [52, 101], to tackle redundant write activities by eliminating a write if its designated PCM cell holds the same value. Then by utilizing such a fine-grained hardware feature, in Cooling-Pool, WAB-FTL 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. Therefore, by using WAB-FTL, a large number of unnecessary write activities in PCM can be avoided. 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.

Based on the same simulation platform adopted by WAP-FTL and PCM-FTL, we conduct a series of experiments on a set of realistic I/O traces collected from notebook and Google Android^{TM}platform. A block-level FTL scheme [93] (denoted by BL-FTL hereinafter) and a page-level FTL scheme *h*FTL [48] are selected as baseline for comparison. The proposed WAB-FTL is compared with BL-FTL and *h*FTL in terms of the total and maximum number of bit flips in each PCM cell with various configurations. Compared with BL-FTL, experimental results show that our WAB-FTL reduces almost half of write activities in PCM, and achieves an average reduction of 80.76% and a maximum reduction of 82.61%

in the maximum number of bit flips. When compared with *h*FTL, experimental results also demonstrate the advantage of our technique in write activities reduction for PCM-based embedded systems.

This chapter makes the following contributions:

- We present for the first time a write-activity-aware block-level flash memory management technique 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 page-level and block-level FTL schemes using a set of realistic I/O workloads collected from notebook and Google Android™2.3.

The rest of this chapter is organized as follows. Section 5.2 introduces the background of system architecture and motivation. Section 5.3 presents our proposed WAB-FTL technique. Section 5.4 reports the experimental results. Finally, we conclude this chapter in Section 5.5.

5.2 Background and Motivation

In this section, we first introduce the architecture of the PCM-based embedded systems. Then we describe the issues of a block-level FTL scheme. Finally, we present the motivation of our work.

5.2.1 PCM-Based Embedded Systems

In this chapter, we target at the PCM-based embedded system with the proposed write-activity-aware block-level flash memory management scheme. As shown in Figure 5.1, a block-level mapping table is maintained by the PCM-based main memory. For reducing write activities of the mapping table in PCM, the proposed WAB-FTL scheme is integrated

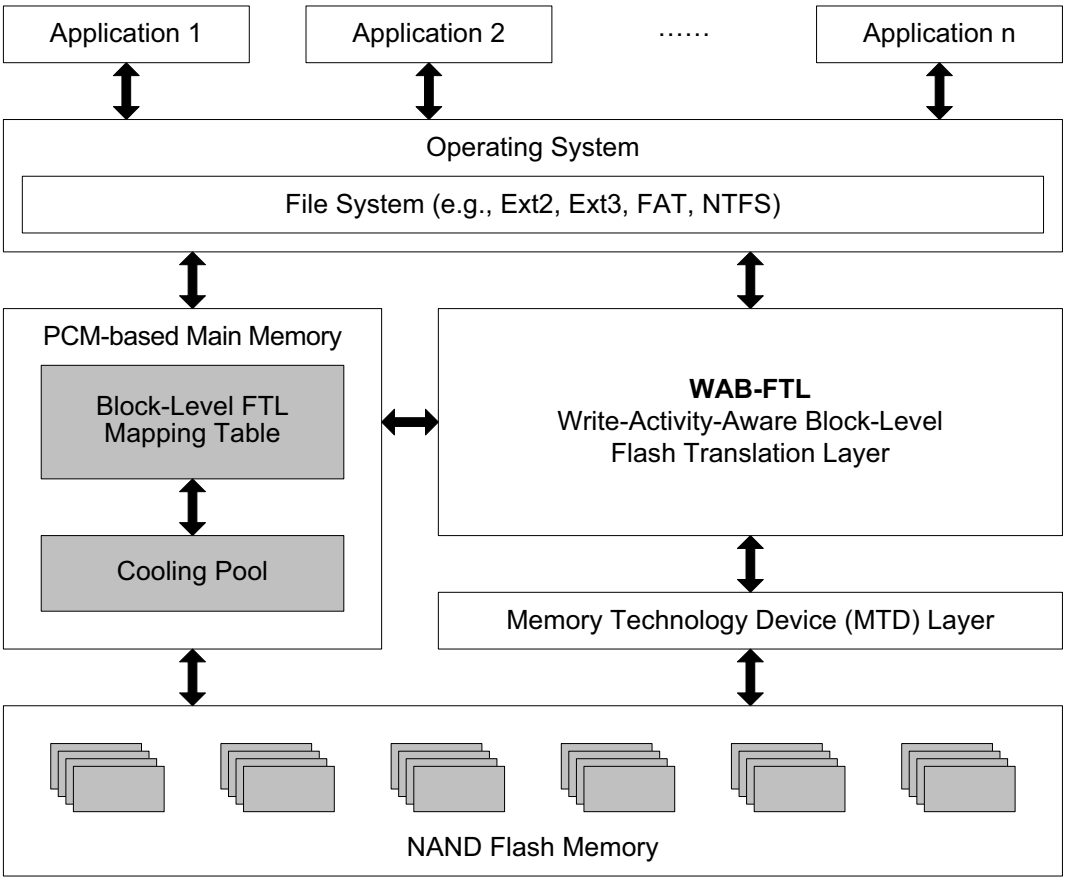


Figure 5.1. PCM-based embedded system with the proposed write-activity-aware block-level flash memory management technique.

into the PCM-based embedded system to replace the original flash translation layer. For the coming read requests, WAB-FTL checks the block-level FTL mapping table in PCM and obtain the corresponding physical page in NAND flash memory for reading. For the coming write requests, WAB-FTL serves the requests by allocating free pages in NAND flash memory and updates the corresponding mapping records of the physical pages in the block-level mapping table in PCM.

5.2.2 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 [93]. 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 Figure 5.2. 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 Figure 5.2(a). According to the I/O requests, Figure 5.2(b) shows the status variation of the blocks in NAND flash memory, and Figure 5.2(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 Figure 5.2(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.

5.2.3 Motivation

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

5.3 WAB-FTL: PCM-Awared Block-Level FTL

In this section, we present the details of our WAB-FTL. We first present an overview of WAB-FTL in Section 5.3.1. We then provide a detailed description of WAB-FTL with Lazy-Merge strategy and Cooling-Pool in Section 5.3.2 and Section 5.3.3, respectively. A wear leveling method of WAB-FTL is presented in Section 5.3.4. Finally, we analyze the proposed WAB-FTL in Section 5.3.5.

5.3.1 Overview of WAB-FTL

The basic idea of WAB-FTL is to preserve each bit in FTL mapping table hosted by PCM from being inverted frequently, e.g., $0 \rightarrow 1 \rightarrow 0$, during the update process of FTL mapping table, such that the maximum number of bit flips in each PCM cell is reduced and the PCM lifetime is enhanced. Thus, to make WAB-FTL write activity aware, we develop the following techniques:

- A new merge strategy, namely Lazy-Merge, is proposed to delay the update of mapping records in FTL mapping table, such that bit flips in PCM are reduced. With Lazy-

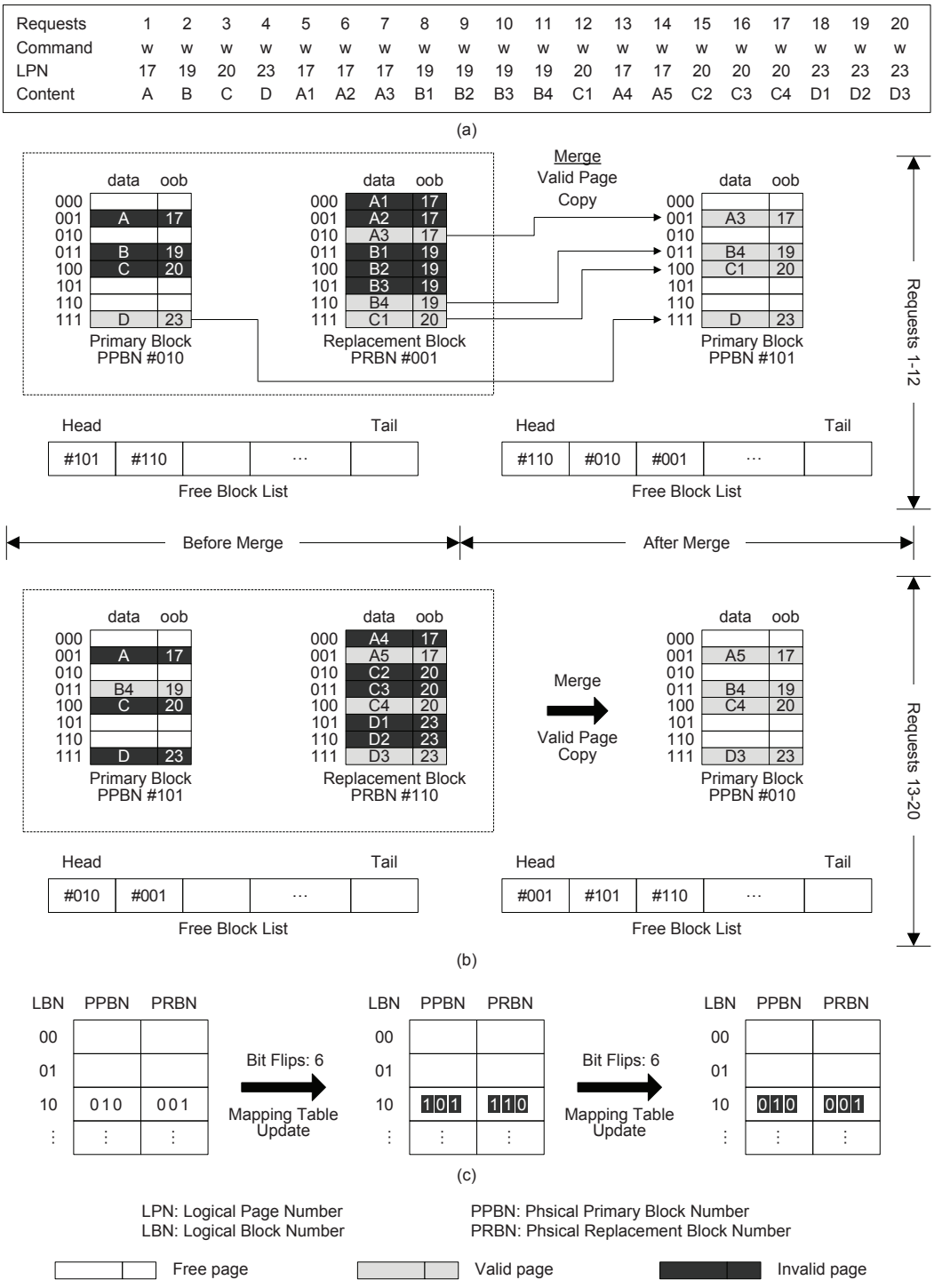


Figure 5.2. Motivational example. (a) I/O access requests. (b) The status variation of blocks in NAND flash memory. (c) The bit flips caused by the update of block-level mapping table in PCM.

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, namely Cooling-Pool, is proposed to reduce write activities in block-level mapping table. As the mapping record of the replacement block is updated more frequently than that of primary block in the block-level mapping table, Cooling-Pool is employed in PCM for caching the frequently updated mapping records to further reduce redundant bit flips in PCM.

5.3.2 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 of block-level mapping table. As mentioned above, in BL-FTL, pages are written consecutively in a replacement block for updated requests, and when it is full, Full-Merge operation will be evoked to erase both the replacement and primary blocks. Then a new primary block is allocated to receive valid pages from the two erased blocks, and the mapping records of the erased blocks in mapping table are updated with the new one. As replacement blocks are always full for handling updated requests, they cannot be used further and have to be erased in merge operation. However, during Full-Merge, the corresponding primary block may not be full, and some free pages can be written by other new write requests later. Therefore, 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 in Figure 5.3. To make the example more understandable, the Cooling-Pool is ignored. Based on the same I/O requests and assumptions in Figure 5.2, for the first 12 requests, the status variation of blocks and mapping table in Figure 5.3 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 Figure 5.3(c), 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 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 5.4 show that our approach can significantly reduce write activities in PCM.

5.3.3 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. Figure 5.4 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

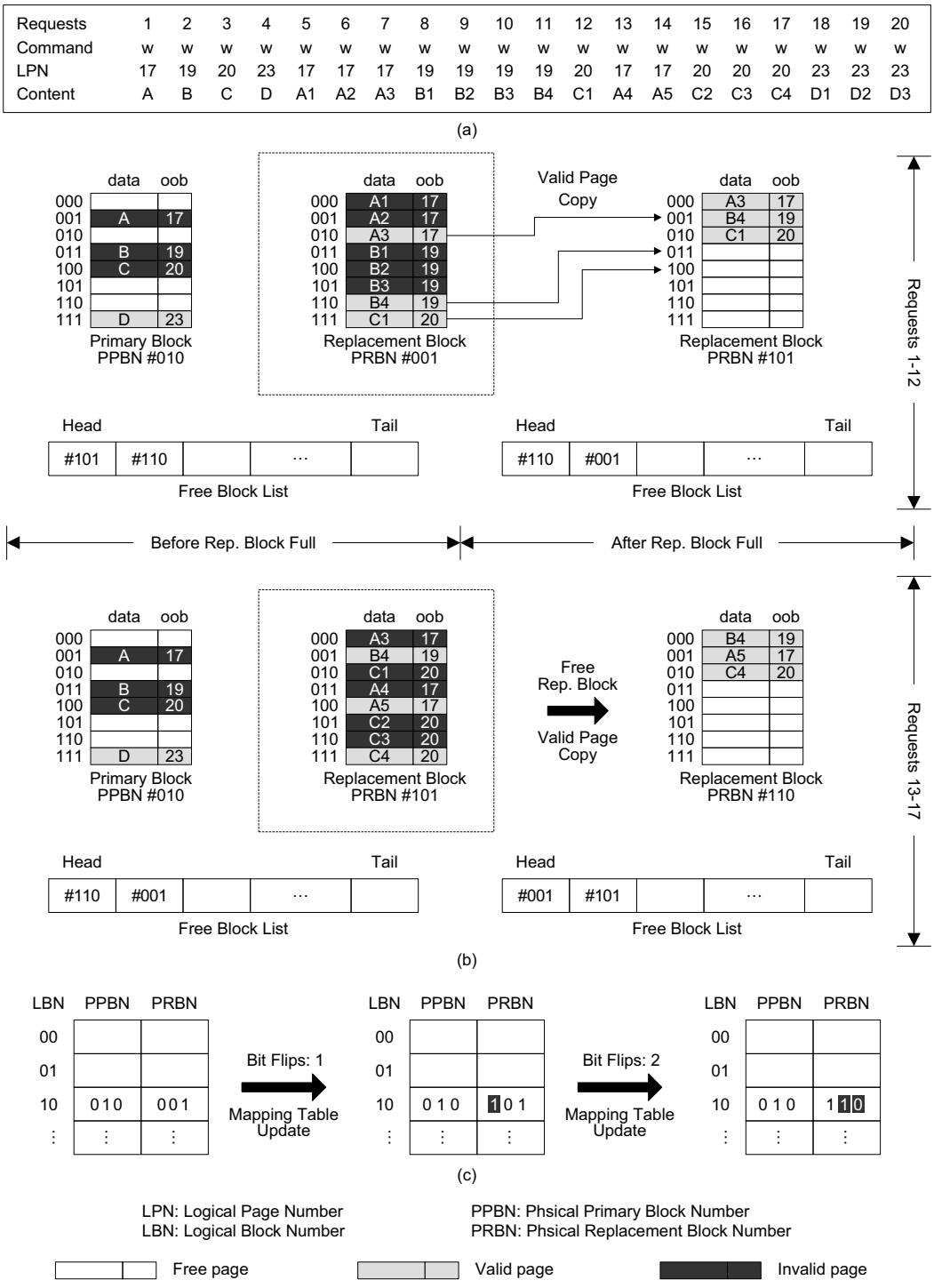


Figure 5.3. Example of WAB-FTL. (a) I/O access requests. (b) The status variation of blocks in NAND flash memory. (c) The bit flips caused by the update of block-level mapping table in PCM.

written into the main mapping table, and the following updated mapping records are written into the Cooling-Pool.

Algorithm 5.3.1 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, $PPBN$) 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.

We assume that the target PCM-based main memory in this chapter has adopted the hardware feature proposed by [52, 101], such that a write is performed in a PCM cell only if the value to be written differs from its original value. So in Cooling-Pool, multiple primary / replacement mapping slots are allocated for comparison between the old mapping records and the new mapping records during mapping table update, to reduce the number of PCM bit flips. For example, as shown in Figure 5.4, an updated mapping record is first written into Pri. Slot #1 and Rep. Slot #1, then the next updated mapping record can be put into Pri./Rep. Slot #2 and the old mapping record in Pri./Rep. slot #1 is invalidated without bits clearance. Therefore, for the following updated mapping records, WAB-FTL actively chooses a destination mapping slot in Cooling-Pool, wherein an old mapping record has a minimum number of Hamming distance with the newly to be updated mapping record. Therefore, in Cooling-Pool, a mapping slot who incurs the minimum number of bit flips will be selected for accommodating the new $PRBN$ value. As a result, the number of bit flips in Cooling-Pool is minimized.

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

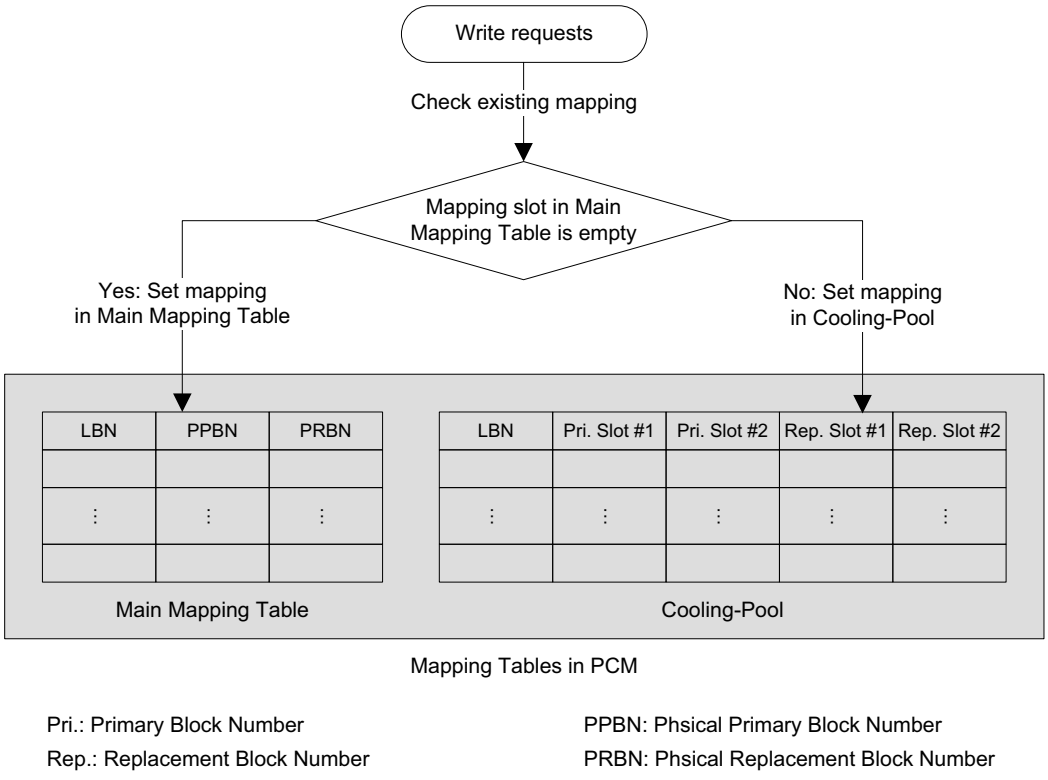


Figure 5.4. WAB-FTL Management.

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.

5.3.4 WAB-FTL Wear Leveling Scheme

In WAB-FTL, as the Cooling-Pool is designed for caching the frequently updated mapping records, so it may become very hot and lead to an uneven distribution of bit flips among all PCM cells. To evenly distribute write activities and enhance PCM endurance, a wear leveling scheme is incorporated into WAB-FTL. Figure 5.5 demonstrates the process of our wear leveling scheme. As shown Figure 5.5(a), the Cooling-Pool becomes hot after

Algorithm 5.3.1 The algorithm of WAB-FTL

Input: I/O requests.

Output: Map LBN to PBN.

Check current mapping state.
if There exists no (LBN, $PPBN$) mapping in main mapping table or Cooling-Pool **then**
    This is a new write, allocate a new primary block $PPBN$, and write the contents based on block offset.
    Add (LBN, $PPBN$) mapping into main mapping table.
end if
if There exists (LBN, $PPBN$) mapping in main mapping table or Cooling-Pool **then**
    This is an update.
    if There exists (LBN, $PRBN$) mapping in main mapping table or Cooling-Pool **then**
        if The number of free pages in replacement block $\leq$ number of update pages to be written **then**
            Allocate one new replacement block.
            if The (LBN, $PRBN$) mapping resides in main mapping table **then**
                Allocate one entry in Cooling-Pool.
            else
                Get (LBN, $PRBN$) entry in Cooling-Pool.
            end if
            Update (LBN, $PRBN$) mapping in the Cooling-Pool entry.
        end if
        Write the new contents to replacement block.
        Invalidate original pages that are updated in primary block (if any) or replacement block (if any).
    else
        Allocate one new replacement block and write the new contents.
        Add (LBN, $PRBN$) mapping into the replacement slot of Cooling-Pool.
    end if
end if

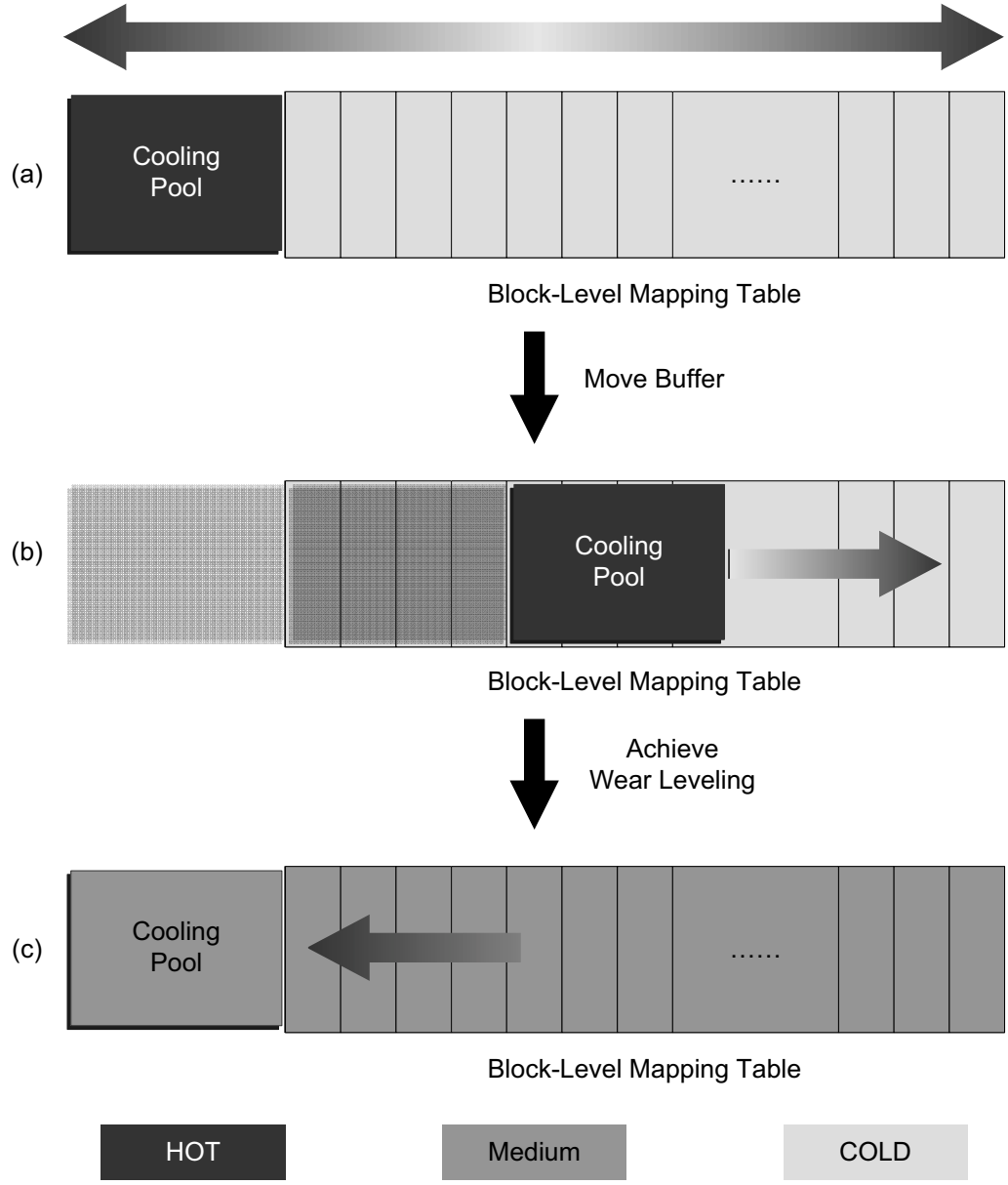


Figure 5.5. Illustration of the wear leveling scheme adopted by WAB-FTL. (a) The initial status of the Cooling-Pool and block-level mapping table with uneven distribution of write activities. (b) Move Cooling-Pool across the whole block-level mapping table region to evenly distribute write activities. (c) Write activities are evenly distribute among the Cooling-Pool and the block-level mapping table after moving Cooling-Pool.

Algorithm 5.3.2 The algorithm of wear leveling in WAB-FTL

Input: The number of writes conducted in PCM cells so far, the allowed number of writes in PCM cells before triggering wear leveling operations.

Output: Next position of the first PCM entry of Cooling-Pool.

```
1: WEAR_LEVELING_THRESHOLD ← The allowed number of writes in PCM cells before triggering wear
   leveling operations.
2: WL_Counter ← The number of writes conducted in PCM cells so far.
3: Current_Offset ← Current position of the first PCM entry of Cooling-Pool.
4: Next_offset ← Next position of the first PCM entry of Cooling-Pool.
5: BTE ← PCM entry of Block-level mapping table.
6: CTE ← PCM entry of Cooling-Pool.
7: if WL_Counter < WEAR_LEVELING_THRESHOLD then
8:     WL_Counter++.
9:     RETURN.
10: else
11:    Next_Offset = (Current_Offset + Cooling_Pool length)%(Total PCM length - Cooling-Pool length).
12:    BTE ← First PCM entry of the block-level mapping table starting from Next_Offset.
13:    CTE ← First PCM entry of Cooling-Pool starting from Current_Offset.
14:    for each PCM entry in Cooling-Pool do
15:        Exchange the content of CTE and BTE.
16:        BTE ← Next PCM entry of the block-level mapping table.
17:        CTE ← Next PCM entry of the Cooling-Pool.
18:    end for
19:    Reset WL_Counter to 0.
20:    RETURN Next_Offset.
21: end if
```

buffering the frequently updated mapping records. However, the block-level mapping table is not so hot as it only serves the infrequently updated mapping records. Therefore, as shown in Figure 5.5(b), during a period of time (e.g., every 2000 I/O requests), the Cooling-Pool is moved across the whole block-level mapping table region in PCM chip. Since the migration of the Cooling-Pool is infrequent, the number of copy operations of mapping records is

acceptable. Finally, in Figure 5.5(b), we can see that the even distribution of write activities (i.e., bit flips) across the whole block-level mapping table region in PCM is obtained. The detailed description of our wear leveling scheme is shown in Algorithm 5.3.2. The experimental results in Section 5.4 confirm the effectiveness of our scheme, and also show that our WAB-FTL can achieve better wear leveling over different traces.

5.3.5 The Analysis of WAB-FTL

We analyze the performance of WAB-FTL over BL-FTL for two extreme cases of write requests, the frequent-update case and the sequential-write case. Given a number of write requests to a NAND flash memory, let N_{wr} be the total number of the write requests. The frequent-update case is used to denote that all write requests are with the same LPN (logical address). Therefore, the content of this LPN will be updated for N_{wr} times. On the contrary, the sequential-write case is used to denote that each of the N_{wr} write requests is with different LPNs. In other words, for the sequential-write case, N_{wr} requests write to N_{wr} distinct pages, and thus no update is needed. In real applications, all the write requests for a NAND flash memory are either one of the two cases or their combination.

Table 5.1 shows the performance analysis of WAB-FTL and BL-FTL over the two extreme cases. In this table, N_p denotes the number of pages in a block; N_{wr} denotes the number of write requests; N_{blk} denotes the number of blocks needed for N_{wr} write requests; N_{allblk} denotes the total number of blocks in NAND flash ($N_{wr} \gg N_{allblk}$); BF_{max} denotes the estimated maximum number of bit flips in a PCM cell. ER_{max} denotes the estimated maximum number of block erase counts. Assume that there are two mapping slots for replacement block in Cooling-Pool and one slot in main mapping table. For frequent-update case, during each one of the N_{wr}/N_p merge operations, only $\log_2^{N_{allblk}}/3$ bit flips occur in WAB-FTL. For BL-FTL, as it will update the mapping records of both primary and replacement blocks, then $2\log_2^{N_{allblk}}$ bit flips occur in each merge operation. For sequential-write case, as the sequential write requests do not trigger update operations, no replacement block is needed. In such case, these two schemes achieve the same results. As mentioned earlier,

Table 5.1. The Performance Comparison of WAB-FTL and BL-FTL.

	Frequent-Update	
	BL-FTL	WAB-FTL
N_{blk}	$\lceil (N_{wr}-1)/N_p \rceil \times 2$	$\lceil (N_{wr}-1)/N_p \rceil + 1$
BF_{max}	$\lfloor (N_{wr}/N_p) \times 2\log_2^{N_{allblk}} \rfloor$	$\lfloor (N_{wr}/N_p) \times (\log_2^{N_{allblk}}/3) \rfloor$
ER_{max}	$\lfloor (N_{wr} \times 2)/(N_{allblk} \times (N_p+1)) \rfloor$	$\lfloor N_{wr}/(N_{allblk} \times (N_p+1)) \rfloor$
	Sequential-Write	
	BL-FTL	WAB-FTL
N_{blk}	$\lceil N_{wr}/N_p \rceil$	$\lceil N_{wr}/N_p \rceil$
BF_{max}	$\lfloor (N_{wr}/N_p) \times \log_2^{N_{allblk}} \rfloor$	$\lfloor (N_{wr}/N_p) \times \log_2^{N_{allblk}} \rfloor$
ER_{max}	$\lfloor N_{wr}/(N_{allblk} \times N_p) \rfloor$	$\lfloor N_{wr}/(N_{allblk} \times N_p) \rfloor$

write requests in real applications are a mix of frequent update and sequential write operations. In most cases, the probability of frequent update operations is much higher than that of sequential write operations [36]. Therefore, WAB-FTL can achieve significant improvement over BL-FTL in terms of the maximum number of bit flips and the maximum number of block erase counts. The experimental results in Section 5.4 depict this fact.

5.4 Evaluation

To evaluate the effectiveness of the proposed 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), and a page-level FTL scheme (*h*FTL), in terms of the maximum and total number of bit flips in PCM cells. Besides, the evaluation for wear leveling is also conducted for WAB-FTL and BL-FTL.

In this chapter, 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.

5.4.1 Experimental Setup

In this chapter, we use the same experimental setup configuration as that given by Table 3.1 in Chapter 3. The evaluation is also conducted through a trace-driven simulation framework, in which a simulator is designed to evaluate WAB-FTL and BL-FTL using a variety of realistic I/O traces collected from notebook and Android platform, respectively. These realistic I/O traces, i.e., CopyFiles, DownFiles, Office, P2P, Communications, Internet, MixedApps, and Multimedia, are introduced in detail in Chapter 3. We will use these eight I/O traces to evaluate our WAB-FTL below.

The simulation framework is shown in Figure 3.6. This simulation framework simulates WAB-FTL management scheme over the PCM-based embedded systems, which consists of a NAND flash memory and a PCM for storing our block-level mapping table. In our experiments, the traces along with various parameters of NAND flash memory, such as block size, page size, etc, are fed into our simulator. The page size, number of pages in a block, and size of the OOB for each page are set as 2KB, 64, and 64 Bytes, respectively. To evaluate our technique, we conduct the experiments on a PCM-based embedded system with 1GB NAND flash memory.

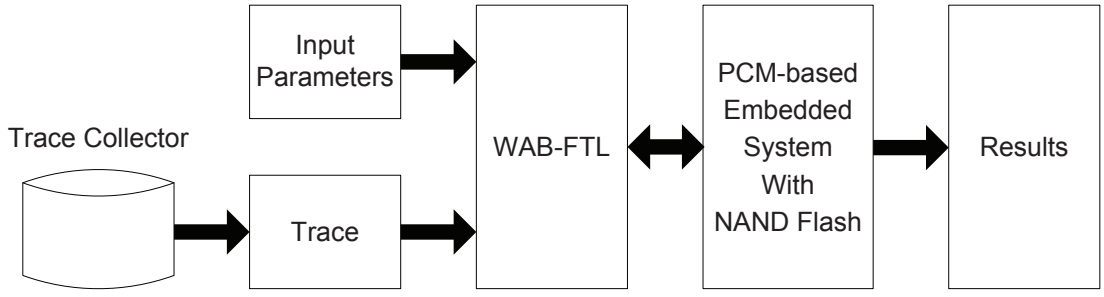


Figure 5.6. The framework of simulation platform for evaluating the proposed block-level WAB-FTL technique.

5.4.2 Metrics

The lifetime of PCM is mainly affected by the worst case of bit flips in a PCM cell, i.e., the maximum number of write operations with different source and destination values in a PCM cell determines the lifetime of PCM. For example, assume that a single PCM cell can sustain at most 10^6 bit flips, then it will wear out if more than 10^6 bit flips occur. Hence for each experiment, we collect both the total and maximum number of bit flips in each PCM cell for each FTL scheme.

5.4.3 Results and Discussion

Based on the above experimental setup and metrics, we present the experimental results with analysis in this section. Impacts on lifetime of both PCM and NAND flash memory are discussed for BL-FTL, *h*FTL [48] and WAB-FTL. Table 5.2 and Table 5.3 summarize the experimental results, which include the maximum and total number of bit flips in PCM cells. We also conduct the wear leveling comparison for WAB-FTL and BL-FTL. Experiments are conducted based on our PCM-based embedded system simulator with 1GB NAND flash memory over eight distinct traces gathered from notebook and Google Android^{TM}platform.

PCM Endurance

In Table 5.2, we observe that WAB-FTL can significantly reduce write activities in PCM in comparison with BL-FTL and *h*FTL. Compared with BL-FTL, WAB-FTL can achieve an average reduction of 83.49% and a maximum reduction of 96.61% in the maximum number of bit flips over the first four DiskMon traces. For Google Android^{TM}traces, WAB-FTL can achieve an average reduction of 65.06% and a maximum reduction of 83.7% in the maximum number of bit flips. Compared with *h*FTL, WAB-FTL can achieve an average reduction of 83.25% and a maximum reduction of 96.34% in the maximum number of bit flips over the first four DiskMon traces. For Google Android^{TM}traces, WAB-FTL can achieve an average reduction of 96.13% and a maximum reduction of 98.84% in the maximum number of bit

Table 5.2. WAB-FTL versus *h*FTL and BL-FTL in terms of the maximum number of bit flips in PCM cells. (1GB NAND flash memory)

Trace Name	*h*FTL	BL-FTL	WAB-FTL	WAB-FTL over *h*FTL	WAB-FTL over BL-FTL
CopyFiles	9977	11079	2953	70.40%	73.35%
DownFiles	21945	23038	2955	86.53%	87.17%
Office	9385	8211	1902	79.73%	76.84%
P2P	74540	80393	2729	96.34%	96.61%
Average				83.25%	83.49%
Communications	158029	16302	3976	97.48%	75.61%
Internet	134281	19548	3449	97.43%	82.36%
MixedApps	139241	9900	1614	98.84%	83.70%
Multimedia	95621	10850	8834	90.76%	18.58%
Average				96.13%	65.06%

flips. As shown, WAB-FTL significantly reduces write activities in PCM.

For the total number of bit flips, shown in Table 5.3, WAB-FTL reduces 16.31% bit flips on average, with a maximum reduction of 17.71% over DiskMon traces, and WAB-FTL reduces 20.14% bit flips on average, with a maximum reduction of 22.15% over Android traces. The write activity reduction is limited as block-level mapping table is small and the write activities are not so high as that of page-level mapping table in *h*FTL. This is confirmed by comparing with *h*FTL, more than 98% of the total number of bit flips are reduced by WAB-FTL for both DiskMon and Android traces. The reason is that page-level mapping table used by *h*FTL is much bigger than block-level mapping table in WAB-FTL, and a series of I/O requests in NAND flash memory may result in one entry update in block-level mapping table while multiple entries update in page-level mapping table. Therefore,

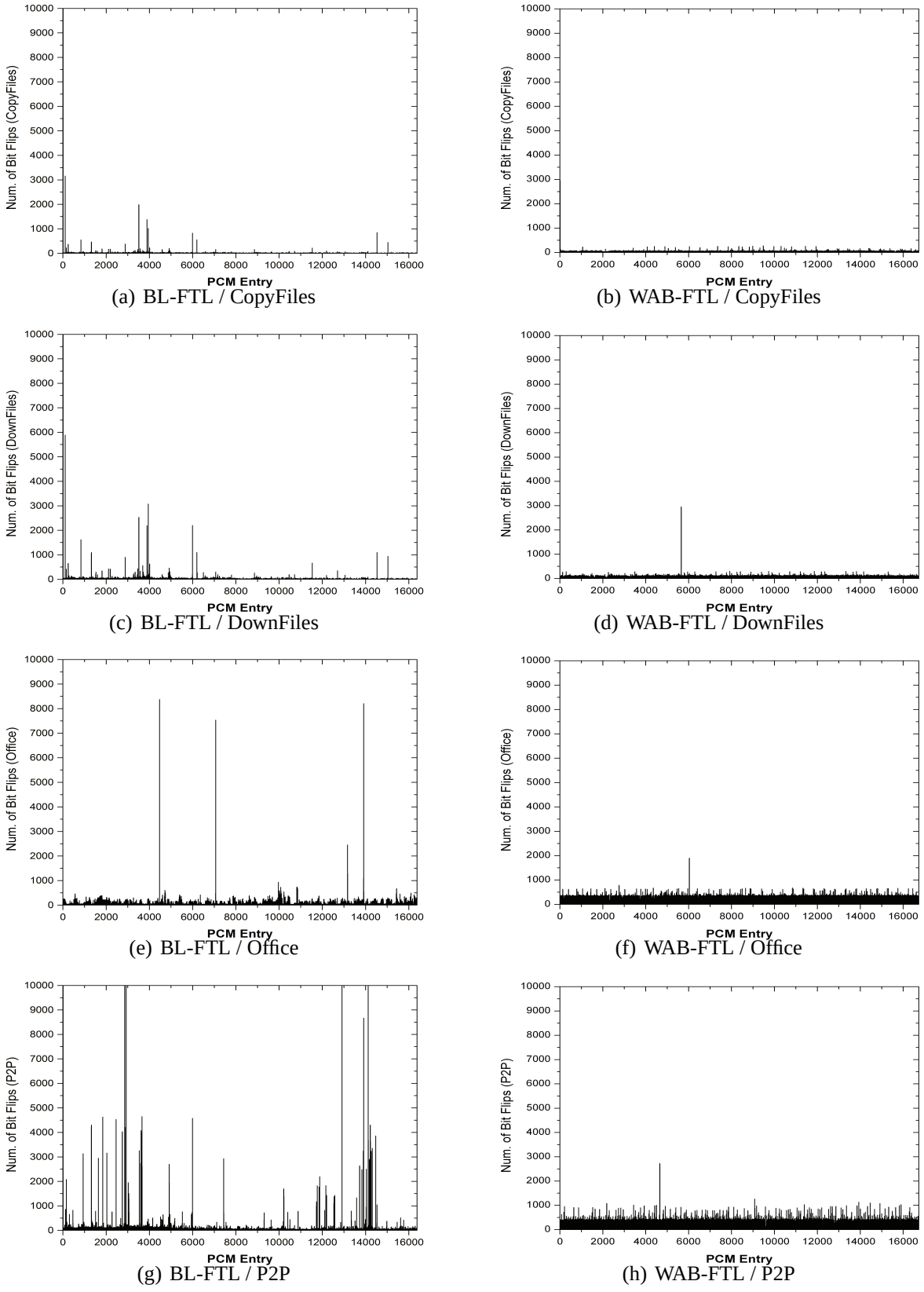
Table 5.3. WAB-FTL versus *h*FTL and BL-FTL in terms of the total number of bit flips in PCM cells. (1GB NAND flash memory)

Trace Name	*h*FTL	BL-FTL	WAB-FTL	WAB-FTL over *h*FTL	WAB-FTL over BL-FTL
CopyFiles	559496658	12963959	10976883	98.04%	15.33%
DownFiles	1756464372	26485707	21796406	98.76%	17.71%
Office	7520028995	91019666	76453444	98.98%	16.00%
P2P	6929967624	98103753	82223776	98.81%	16.19%
Average				98.65%	16.31%
Communications	9883730134	147743017	118227997	98.80%	19.98%
Internet	9555187107	145098438	114409247	98.80%	21.15%
MixedApps	6591084832	99759148	77663514	98.82%	22.15%
Multimedia	8347917692	133167616	110145808	98.68%	17.29%
Average				98.78%	20.14%

WAB-FTL is more applicable in PCM-based embedded systems. By adopting WAB-FTL, the lifetime of PCM can be prolonged.

PCM Wear Leveling

Figure 5.7 and Figure 5.8 show the distribution of the maximum number of bit flips among all mapping table entries in a PCM-based embedded system with 1GB NAND flash memory over the DiskMon traces and Google Android™traces, respectively. For each sub-figure, the x-axis shows the total number of mapping entries inside the block-level mapping table and Cooling-Pool in PCM, and the y-axis shows the maximum number of bit flips extracted from each mapping entry of the block-level mapping table and Cooling-Pool. To present the distributions clearly, we restrict the maximum number of bit flips on y-axis to 10,000.



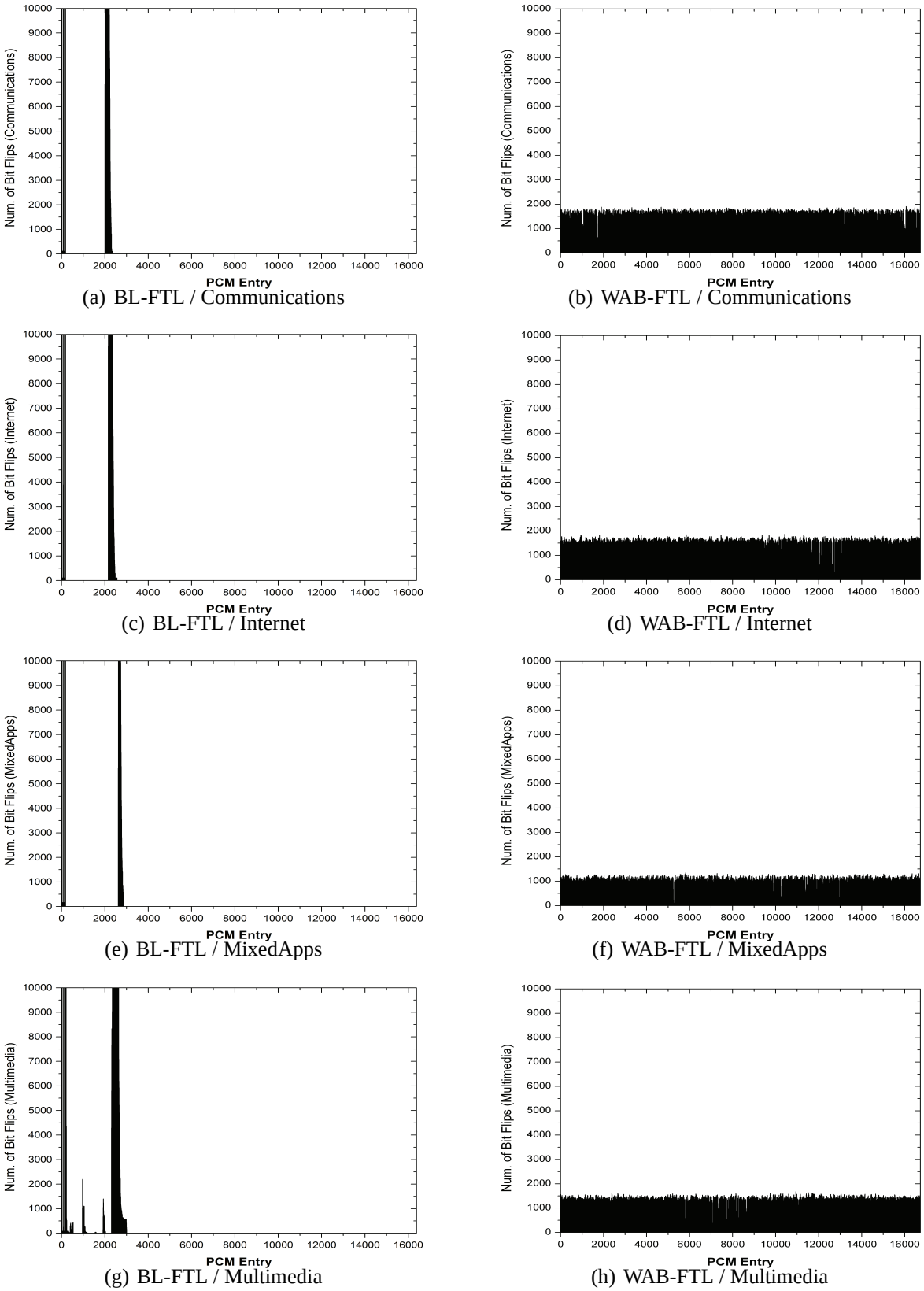
(a) BL-FTL / CopyFiles (b) WAB-FTL / CopyFiles

(c) BL-FTL / DownFiles (d) WAB-FTL / DownFiles

(e) BL-FTL / Office (f) WAB-FTL / Office

(g) BL-FTL / P2P (h) WAB-FTL / P2P

Figure 5.7. The wear leveling comparison of BL-FTL and WAB-FTL in a PCM-based embedded system with 1GB NAND flash memory over four traces collected by DiskMon.



(a) BL-FTL / Communications (b) WAB-FTL / Communications

(c) BL-FTL / Internet (d) WAB-FTL / Internet

(e) BL-FTL / MixedApps (f) WAB-FTL / MixedApps

(g) BL-FTL / Multimedia (h) WAB-FTL / Multimedia

Figure 5.8. The wear leveling comparison of BL-FTL and WAB-FTL in a PCM-based embedded system with 1GB NAND flash memory over four traces collected by Android.

As shown in Figure 5.7, for BL-FTL scheme, we observe that the distribution of the maximum number of bit flips varies a lot, and this may impose a fast worn-out of PCM cells. Compared with BL-FTL, by adopting our wear leveling scheme described in Section 5.3.4, WAB-FTL distributes the maximum number of bit flips more evenly among all PCM cells. In Figure 5.8, we can see that the write activities of BL-FTL are mapped to a specific region in PCM due to access pattern of I/O requests, and the maximum number of bit flips in most entries is greater than 10,000. On the contrary, WAB-FTL evenly distributes the maximum number of bit flips across PCM chip for the Google Android™traces, and almost all of them are below 2,000. In summary, WAB-FTL delivers dramatically better reliability than the baseline scheme.

5.5 Summary

In this chapter, we proposed a write-activity-aware block-level flash memory management technique, 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 notebook and Google Android™2.3. Experimental results show that our technique significantly reduces write activities in PCM, and achieves an average reduction of 83.49% (65.06%) and a maximum reduction of 96.61% (83.7%) in the maximum number of bit flips in comparison with a well-known block-level FTL scheme over DiskMon traces (Android traces). In addition, we also demonstrate the advantage of WAB-FTL in write activity reduction when compared with a page-level FTL scheme.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

PCM has been used in embedded systems, so effective management schemes are needed to explore its advantages and solve the existing problems. This thesis studied the PCM-based embedded systems with NAND flash memory. To the best of our knowledge, this is the first work to study how to effectively manage NAND flash memory while enhancing the lifetime of PCM at software-level. We hope this work can serve as a first step towards the design of write-activity-aware FTL for the PCM-based embedded systems via simple and feasible modifications.

As the traditional flash memory management techniques impose significant write activities in PCM-based main memory, it is necessary to redesign the flash memory management technique for PCM-based embedded systems. Therefore, in this thesis, three write-activity-aware flash memory management techniques are presented to improve the lifetime and performance of the PCM-based embedded systems.

- First, we have proposed a write-activity-aware page-level flash memory management technique, named WAP-FTL, to exploit the advantages of the well-known FTL implementations in order to reduce write activities in PCM for enhancing lifetime of the PCM-based embedded systems. In our WAP-FTL, the write activity reduction is achieved by preserving each bit in page-level FTL mapping table that hosted by PCM from being inverted frequently. Unlike the traditional page-level FTL scheme [48], WAP-FTL can actively choose a physical page whose physical address incurs the min-

imum number of bit flips in FTL page-level mapping table hosted by PCM, so as to effectively reduce write activities in PCM cells. However, with a set of real-life workloads, the experimental results show that our WAP-FTL technique cannot fully reduce write activities compared to the baseline scheme *h*FTL, especially after garbage collection happens. The reason is that WAP-FTL does not consider the behavior of I/O requests, which are mixed with sequential and random requests, and write-activity-aware strategy may introduce extra valid page copy overhead in garbage collection. These observations motivate us to further extend this work for reducing write activities in PCM-based embedded systems.

- Second, we have proposed a write-activity-aware two-level flash memory management technique, named PCM-FTL, which takes the first step to significantly reduce write activities in PCM-based embedded systems with NAND flash memory. In our PCM-FTL, the performance improvement is achieved by preserving a bit in a PCM cell from being inverted frequently. Through a two-level mapping mechanism and a write-activity-aware strategy, unnecessary write activities in PCM are directly reduced. We conducted experiments on a set of realistic I/O workload collected by DiskMon and Google Android. For the DiskMon traces, 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 in a PCM-based embedded system with 1GB (4GB) NAND flash memory. For the Google Android traces, the experimental results show that the maximum number of bit flips among PCM cells can be reduced by 99.82% (99.86%) on average, and the total number of bit flips of all PCM cells can be reduced by 93.10% (98.17%) on average in a PCM-based embedded system with 1GB (4GB) NAND flash memory. Furthermore, the results show that PCM-FTL can evenly distribute write activities among all PCM cells in comparison with a representative baseline FTL scheme.

- Third, we have proposed a write-activity-aware block-level flash memory management technique, WAB-FTL, which can effectively reduce write activities in PCM-based em-

bedded 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 notebook and Google Android^{TM}2.3. Experimental results show that our technique significantly reduces write activities in PCM, and achieves an average reduction of 83.49% (65.06%) and a maximum reduction of 96.61% (83.7%) in the maximum number of bit flips in comparison with a well-known block-level FTL scheme over DiskMon traces (Android traces). In addition, we also demonstrate the advantage of WAB-FTL in write activity reduction when compared with a page-level FTL scheme.

6.2 Future Work

The work presented in this thesis can be extended in different directions in the future.

- First, energy and thermal issues of PCM are not studied in this thesis. However, our proposed techniques could significantly reduce the total number of write activities which is related to the energy consumption and thermal dissipation in PCM. Therefore, we will investigate the energy consumption and thermal issues of PCM to design an energy- or thermal-aware scheme to improve the performance and reliability of PCM-based embedded systems.

- Second, currently our approach is based on the single-level cell (SLC) PCM. Compared to SLC PCM, MLC PCM can provide bigger capacity by storing more than one bit information per cell. However, as the resistance margin between two adjacent states in MLC PCM becomes smaller, its states are vulnerable to be changed by various factors such as process variation, resistance drift, and the thermal disturbance from vicinity reads/writes. Therefore, we will extend our approach to MLC PCM and propose schemes that can handle the issues of MLC PCM.

- Third, this work only focuses on reducing write activities of FTL mapping table in PCM. However, in realistic systems, some metadata and code may also be stored into PCM. The access pattern of these information and mapping table may vary a lot, so it is also interesting to develop techniques to effectively manage different types of information in PCM and evenly distribute writes across these information for enhancing PCM lifetime.

- Fourth, the techniques proposed in this thesis mainly corresponds to the page-level and block-level FTL designs, and we can further explore the possibility of making some hybrid-level FTL designs write activity aware for PCM-based embedded systems.

- Finally, the existing FTL schemes are originally designed for a DRAM-based main memory with NAND flash memory, and they do not consider the distinct feature of PCM, so a possible research direction is to propose a light-weight translation layer that is specially designed for PCM-based embedded systems.

REFERENCES

[1] DiskMon for Windows. *http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx*.

[2] Memory technology device (MTD) subsystem for linux. *http://www.linux-mtd.infradead.org/*, 2009.

[3] S.J. Ahn, Y.J. Song, C.W. Jeong, J.M. Shin, Y. Fai, Y.N. Hwang, S.H. Lee, K.C. Ryoo, S.Y. Lee, J.H. Park, H. Horii, Y.H. Ha, J.H. Yi, B.J. Kuh, G.H. Koh, G.T. Jeong, H.S. Jeong, Kinam Kim, and B.I. Ryu. Highly manufacturable high density phase change memory of 64Mb and beyond. In *2004 IEEE International Electron Devices Meeting (IEDM '04)*, pages 907–910, December 2004.

[4] S.J. Ahn, Y.J. Song, C.W. Jeong, J.M. Shin, Y. Fai, Y.N. Hwang, S.H. Lee, K.C. Ryoo, S.Y. Lee, J.H. Park, H. Horii, Y.H. Ha, J.H. Yi, B.J. Kuh, G.H. Koh, G.T. Jeong, H.S. Jeong, Kinam Kim, and B.I. Ryu. Highly manufacturable high density phase change memory of 64Mb and beyond. In *IEEE International Electron Devices Meeting 2004 (IEDM '04)*, pages 907–910, December 2004.

[5] Amir Ban. Flash file system. *US patent 5,404,485*, April 1995.

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

[7] F. Bedeschi, R. Fackenthal, C. Resta, E.M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. Chow, A. Cabrini, G.M.A. Calvi, R. Faravelli, A. Fantini, G. Torelli, Duane Mills, R. Gastaldi, and G. Casagrande. A multi-level-cell bipolar-selected phase-

change memory. In *2008 IEEE International Solid-State Circuits Conference (ISSCC '08)*, pages 428–625, February 2008.

[8] S. Bock, B. Childers, R. Melhem, D. Mosse and, and Youtao Zhang. Analyzing the impact of useless write-backs on the endurance and energy consumption of PCM main memory. In *2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '11)*, pages 56–65, April 2011.

[9] Li-Pin Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*, pages 1126–1130, 2007.

[10] Li-Pin Chang and Tei-Wei Kuo. A Real-Time garbage collection mechanism for flash-memory storage systems in embedded systems. In *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications (RTCSA '02)*, March 2002.

[11] Li-Pin Chang and Tei-Wei Kuo. An efficient management scheme for large-scale flash-memory storage systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*, pages 862–868, 2004.

[12] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. 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)*, pages 212–217, 2007.

[13] Yuan-Hao Chang and Tei-Wei Kuo. A commitment-based management strategy for the performance and reliability enhancement of flash-memory storage systems. In *Proceedings of the 46th Annual Design Automation Conference (DAC '09)*, pages 858–863, 2009.

[14] Siddhartha Chhabra and Yan Solihin. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *Proceeding of the 38th annual international symposium on Computer architecture (ISCA '11)*, pages 177–188, 2011.

[15] Hyunjin Cho, Dongkun Shin, and Young Ik Eom. KAST: K-associative sector translation for NAND flash memory in Real-Time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09)*, pages 507–512, 2009.

[16] Sangyeun Cho and Hyunjin Lee. 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)*, pages 347–357, 2009.

[17] Siddharth Choudhuri and Tony Givargis. Performance improvement of block based NAND flash translation layer. In *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pages 257–262, 2007.

[18] Yuan-Sheng Chu, Jen-Wei Hsieh, Yuan-Hao Chang, and Tei-Wei Kuo. A set-based mapping strategy for flash-memory reliability enhancement. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09)*, pages 405–410, 2009.

[19] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. A survey of flash translation layer. *Journal of Systems Architecture*, 55(5-6):332–343, 2009.

[20] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09)*, pages 133–146, 2009.

[21] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. PDRAM: a hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Annual Design Automation Conference (DAC '09)*, pages 664–469, 2009.

[22] Jianbo Dong, Lei Zhang, Yinhe Han, Ying Wang, and Xiaowei Li. Wear rate leveling: Lifetime enhancement of PRAM with endurance variation. In *2011 48th*

ACM/EDAC/IEEE Design Automation Conference (DAC '11), pages 972–977, June 2011.

[23] Xiangyu Dong and Yuan Xie. AdaMS: Adaptive MLC/SLC phase-change memory design for file storage. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 31–36, January 2011.

[24] S. Eilert, M. Leinwander, and G. Crisenza. Phase change memory: A new memory enables new memory usage models. In *IEEE International Memory Workshop (IMW '09)*, pages 1–2, May 2009.

[25] S. Eilert, M. Leinwander, and G. Crisenza. Phase change memory: A new memory enables new memory usage models. In *IEEE International Memory Workshop (IMW '09)*, pages 1–2, May 2009.

[26] Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé. Increasing PCM main memory lifetime. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*, pages 914–919, 2010.

[27] A.P. Ferreira, B. Childers, R. Melhem, D. Mosse, and M. Yousif. Using PCM in next-generation embedded space applications. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '10)*, pages 153–162, April 2010.

[28] Google Corporation. Google Android. *http://www.android.com*, 2011.

[29] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, pages 229–240, 2009.

[30] S. Hanzawa, N. Kitai, K. Osada, A. Kotabe, Y. Matsui, N. Matsuzaki, N. Takaura, M. Moniwa, and T. Kawahara. A 512kB embedded phase change memory with

416kB/s write throughput at 100μA cell write current. In *2007 IEEE International Solid-State Circuits Conference (ISSCC '07)*, pages 474–616, February 2007.

[31] Jen-Wei Hsieh, Tei-Wei Kuo, Po-Liang Wu, and Yu-Chung Huang. Energy-efficient and performance-enhanced disks using flash-memory cache. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED '07)*, pages 334–339, 2007.

[32] Jen-Wei Hsieh, Yi-Lin Tsai, Tei-Wei Kuo, and Tzao-Lin Lee. Configurable flash-memory management: Performance versus overheads. *IEEE Transactions on Computers*, 57(11):1571–1583, November 2008.

[33] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Yi He, Meikang Qiu, and Edwin H.-M. Sha. 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)*, pages 350–355, 2010.

[34] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Qingfeng Zhuge, and Edwin H.-M. Sha. Minimizing write activities to non-volatile memory via scheduling and recomputation. In *Proceedings of the 2010 IEEE Symposium on Application Specific Processors (SASP '10)*, 2010.

[35] Po-Chun Huang, Yuan-Hao Chang, Tei-Wei Kuo, Jen-Wei Hsieh, and Miller Lin. The behavior analysis of flash-memory storage systems. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC '08)*, pages 529–534, 2008.

[36] Po-Chun Huang, Yuan-Hao Chang, Tei-Wei Kuo, Jen-Wei Hsieh, and Miller Lin. The behavior analysis of flash-memory storage systems. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC '08)*, pages 529–534, 2008.

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

[38] ITRS. International technology roadmap for semiconductors (2009 edition). *http://www.itrs.net*, 2009.

[39] Lei Jiang, Youtao Zhang, and Jun Yang. Enhancing phase change memory lifetime through fine-grained current regulation and voltage upscaling. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design (ISLPED '11)*, pages 127–132, 2011.

[40] Yongsoo Joo, Youngjin Cho, Naehyuck Chang, and Donghwa Shin. Energy-aware data compression for multi-level cell (MLC) flash memory. In *Proceedings of the Annual Conference on Design Automation (DAC '07)*, pages 716–719, June 2007.

[41] Yongsoo Joo, Yongseok Choi, Chanik park, Sung Woo Chung, Eui-Young Chung, and Naehyuck Chang. Demand paging for OneNAND flash eXecute-In-Place. In *Proceedings of the 4th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06)*, pages 229–234, October 2006.

[42] Yongsoo Joo, Yongseok Choi, Jaehyun Park, Chanik park, Sung Woo Chung, Eui-Young Chung, and Naehyuck Chang. Energy and performance optimization of demand paging with OneNAND flash. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):1969–1982, November 2008.

[43] Yongsoo Joo, Dimin Niu, Xiangyu Dong, Guangyu Sun, Naehyuck Chang, and Yuan Xie. Energy- and endurance-aware design of phase change memory caches. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*, pages 136–141, 2010.

[44] Dawoon Jung, Yoon-Hee Chae, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '07)*, pages 160–164, 2007.

[45] D-H. Kang, J.-H. Lee, J.H. Kong, D. Ha, J. Yu, C.Y. Um, J.H. Park, F. Yeung, J.H. Kim, W.I. Park, Y.J. Jeon, M.K. Lee, Y.J. Song, J.H. Oh, G.T. Jeong, and H.S. Jeong.

Two-bit cell operation in diode-switch phase change memory cells with 90nm technology. In *2008 IEEE Symposium on VLSI Technology*, pages 98–99, June 2008.

[46] DerChang Kau, S. Tang, I.V. Karpov, R. Dodge, B. Klehn, J.A. Kalb, J. Strand, A. Diaz, N. Leung, J. Wu, S. Lee, T. Langtry, Kuo wei Chang, C. Papagianni, Jinwook Lee, J. Hirst, S. Erra, E. Flores, N. Righos, H. Castro, and G. Spadini. A stackable cross point phase change memory. In *2009 IEEE International Electron Devices Meeting (IEDM '09)*, pages 1–4, December 2009.

[47] Jesung Kim, Jong Min Kim, S.H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, May 2002.

[48] Jin Kyu Kim, Hyung Gyu Lee, Shinho Choi, and Kyoung Il Bahng. A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In *Proceedings of EMSOFT '08*, pages 31–40, 2008.

[49] Jingfei Kong and Huiyang Zhou. Improving privacy and lifetime of PCM-based main memory. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '10)*, pages 333–342, July 2010.

[50] R. Krishnamurthy. Comparing samsung nor-compatible pcm with nor. 2011.

[51] Tei-Wei Kuo, Yuan-Hao Chang, Po-Chun Huang, and Che-Wei Chang. Special issues in flash. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '08)*, pages 821–826, November 2008.

[52] B.C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-change technology and the future of main memory. *IEEE Micro*, 30(1):143–143, January 2010.

[53] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09)*, pages 2–13, 2009.

[54] Jongmin Lee, Sunghoon Kim, Hunki Kwon, Choulseung Hyun, Seongjun Ahn, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Block recycling schemes and their cost-based optimization in NAND flash memory based storage system. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT '07)*, pages 174–182, 2007.

[55] Kwang-Jin Lee, Beak-Hyung Cho, Woo-Yeong Cho, Sangbeom Kang, Byung-Gil Choi, Hyung-Rok Oh, Chang-Soo Lee, Hye-Jin Kim, Joon-Min Park, Qi Wang, Mu-Hui Park, Yu-Hwan Ro, Joon-Yong Choi, Ki-Sung Kim, Young-Ran Kim, In-Cheol Shin, Ki-Won Lim, Ho-Keun Cho, Chang-Han Choi, Won-Ryul Chung, Du-Eung Kim, Kwang-Suk Yu, Gi-Tae Jeong, Hong-Sik Jeong, Choong-Keun Kwak, Chang-Hyun Kim, and Kinam Kim. A 90nm 1.8V 512Mb diode-switch PRAM with 266MB/s read throughput. In *2007 IEEE International Solid-State Circuits Conference (ISSCC '07)*, pages 472–616, February 2007.

[56] Kwangyoon Lee and Alex Orailoglu. Application specific low latency instruction cache for NAND flash memory based embedded systems. In *Proceedings of the 2008 IEEE Symposium on Application Specific Processors (SASP '08)*, pages 69–74, June 2008.

[57] Kwangyoon Lee and Alex Orailoglu. Application specific non-volatile primary memory for embedded systems. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '08)*, pages 31–36, 2008.

[58] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems*, 6(3):18, 2007.

[59] Yu Li, Jianliang Xu, Byron Choi, and Haibo Hu. StableBuffer: optimizing write performance for DBMS applications on flash devices. In *Proceedings of the 19th*

ACM international conference on Information and knowledge management (CIKM '10), pages 339–348, 2010.

[60] D. Liu, Y. Wang, Z. Qin, Z. Shao, and Y. Guan. A space reuse strategy for flash translation layers in SLC NAND flash memory storage systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–14, 2011.

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

[62] T. Nirschl, J.B. Phipp, T.D. Happ, G.W. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y.C. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam. Write strategies for 2 and 4-bit multi-level phase-change memory. In *2007 IEEE International Electron Devices Meeting (IEDM '07)*, pages 461–464, December 2007.

[63] Sai Tung On, Yinan Li, Bingsheng He, Ming Wu, Qiong Luo, and Jianliang Xu. FD-buffer: a buffer manager for databases on flash disks. In *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM '10)*, pages 1297–1300, 2010.

[64] Stanford R. Ovshinsky. Symmetrical current controlling device. *US patent 3271591*, September 1966.

[65] Stanford R. Ovshinsky. Reversible electrical switching phenomena in disordered structures. *Physical Review Letters*, 21:1450–1453, November 1968.

[66] N. Papandreou, H. Pozidis, T. Mittelholzer, G.F. Close, M. Breitwisch, C. Lam, and E. Eleftheriou. Drift-tolerant multilevel phase-change memory. In *2011 3rd IEEE International Memory Workshop (IMW),*, pages 1–4, May 2011.

[67] Chanik Park, Wonmoon Cheon, Jeonguk Kang, Kangho Roh, Wonhee Cho, and Jin-Soo Kim. A reconfigurable FTL (flash translation layer) architecture for NAND flash-

based applications. *ACM Transactions on Embedded Computing Systems*, 7(4):1–23, 2008.

[68] Youngwoo Park and Kyu Ho Park. High-performance scalable flash file system using virtual metadata storage with phase-change RAM. *IEEE Transactions on Computers*, 60(3):321–334, Mar. 2011.

[69] Youngwoo Park and Kyu Ho Park. High-performance scalable flash file system using virtual metadata storage with phase-change ram. *IEEE Transactions on Computers*, 60(3):321–334, March 2011.

[70] Zhiwei Qin, Yi Wang, Duo Liu, and Zili Shao. Demand-based block-level address mapping in large-scale NAND flash storage systems. In *Proceedings of the 8th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '10)*, 2010. (to appear).

[71] Zhiwei Qin, Yi Wang, Duo Liu, and Zili Shao. 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)*, pages 173–182, 2010.

[72] Zhiwei Qin, Yi Wang, Duo Liu, and Zili Shao. 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)*, pages 157–166, Apr. 2011.

[73] Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. MNFTL: an efficient flash translation layer for MLC NAND flash memory storage systems. In *Proceedings of the 48th Design Automation Conference (DAC '11)*, pages 17–22, Jun. 2011.

[74] M.K. Qureshi, M.M. Franceschini, and L.A. Lastras-Montano. Improving read performance of phase change memories via write cancellation and write pausing. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA '10)*, pages 1 –11, January 2010.

[75] M.K. Qureshi, A. Seznec, L.A. Lastras, and M.M. Franceschini. Practical and secure PCM systems by online detection of malicious write streams. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture (HPCA '11)*, pages 478–489, February 2011.

[76] Moinuddin K. Qureshi, Michele M. Franceschini, Luis A. Lastras-Montaño, and John P. Karidis. Morphable memory system: a robust architecture for exploiting multi-level phase change memories. In *Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10)*, pages 153–162, 2010.

[77] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 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)*, pages 14–23, 2009.

[78] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09)*, pages 24–33, 2009.

[79] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. c. Chen, R. M. Shelby, M. Salinga, D. Krebs, S. h. Chen, H. l. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 2008.

[80] Simone Raoux, Charles T. Rettner, Jean L. Jordan-Sweet, Andrew J. Kellock, Teya Topuria, Philip M. Rice, and Dolores C. Miller. Direct observation of amorphous to crystalline phase transitions in nanoparticle arrays of phase change materials. *Journal of Applied Physics*, 102(9):094305–094308, November 2007.

[81] Samsung. Samsung ships industrys first multi-chip package with a PRAM chip for handsets. *http://www.samsung.com/us/business/semiconductor/newsView.do?news*

id=1149, April 2010.

[82] SAMSUNG Corporation. SAMSUNG NAND flash. *http://www.samsung.com/global/business/semiconductor*, 2009.

[83] Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10)*, pages 383–394, 2010.

[84] G. Servalli. A 45nm generation phase change memory technology. In *2009 IEEE International Electron Devices Meeting (IEDM '09)*, pages 1–4, December 2009.

[85] A. Seznec. A phase change memory as a secure main memory. *Computer Architecture Letters*, 9(1):5–8, January 2010.

[86] A. Shah. Samsung to put pcm for smartphones in chip package. *PCWorld*, 2010.

[87] Guangyu Sun, Yongsoo Joo, Yibo Chen, Dimin Niu, Yuan Xie, Yiran Chen, and Hai Li. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA '10)*, pages 1–12, Jan. 2010.

[88] Guangyu Sun, Dimin Niu, Jin Ouyang, and Yuan Xie. A frequent-value based PRAM memory architecture. In *Proceedings of the 2011 Conference on Asia and South Pacific Design Automation (ASP-DAC '11)*, pages 211–216, 2011.

[89] Yi-Lin Tsai, Jen-Wei Hsieh, and Tei-Wei Kuo. Configurable NAND flash translation layer. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*, pages 118–127, 2006.

[90] Yi Wang, Duo Liu, Zhiwei Qin, and Zili Shao. 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)*, pages 1–6, Mar. 2011.

[91] Yi Wang, Duo Liu, Meng Wang, Zhiwei Qin, Zili Shao, and Yong Guan. 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)*, pages 163–172, 2010.

[92] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo. An efficient B-Tree layer for flash-memory storage systems. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA '03)*, pages 409–430, February 2003.

[93] Chin-Hsien Wu and Tei-Wei Kuo. 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)*, pages 601–606, 2006.

[94] Chin-Hsien Wu, Tei-Wei Kuo, and Chia-Lin Yang. A space-efficient caching mechanism for flash-memory address translation. In *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC '06)*, pages 64–71, 2006.

[95] Po-Liang Wu, Yuan-Hao Chang, and Tei-Wei Kuo. A file-system-aware FTL design for flash-memory storage systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09)*, pages 393–398, 2009.

[96] Yuan Xie. Modeling, architecture, and applications for emerging memory technologies. *IEEE Design Test of Computers*, 28(1):44–51, Jan.–Feb. 2011.

[97] Noboru Yamada, Eiji Ohno, Kenichi Nishiuchi, Nobuo Akahira, and Masatoshi Takao. Rapid-phase transitions of GeTe-Sb2Te3 pseudobinary amorphous thin films for an optical disk memory. *Journal of Applied Physics*, 69(5):2849–2856, March 1991.

[98] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael Harding, and Onur Mutlu. Row buffer locality-aware data placement in hybrid memories. *SAFARI Technical Report No. 2011-005*, September 2011.

[99] Wangyuan Zhang and Tao Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '42)*, pages 2–13, 2009.

[100] Wangyuan Zhang and Tao Li. Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN '11)*, pages 197–208, June 2011.

[101] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 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)*, pages 14–23, 2009.