



國立臺灣科技大學  
電子工程系

---

## 碩士學位論文

善用固態硬碟之多控制器的平行處理能力

Exploiting Multi-controller Parallelism for  
Solid-State Drives

宋鴻邑

M10002137

指導教授：吳晉賢博士

中華民國一百零二年十一月十一日



M10002137



## 碩士學位論文指導教授推薦書

本校 電子工程系 宋鴻邑(SUNG, HUNG-YI) 君

所提之論文：

善用固態硬碟之多控制器的平行處理能力

係由本人指導撰述，同意提付審查。

指導教授：吳晉賢

指導教授

吳晉賢

102年 11 月 11 日



# 碩士學位考試委員審定書



M10002137

指導教授：吳晉賢

本校 電子工程系 宋鴻邑 君

所提之論文：

善用固態硬碟之多控制器的平行處理能力

經本委員會審定通過，特此證明。

學校考試委員會

委員：

林昌明

陳維志

阮聖新

林淵翔

吳晉賢

指導教授：

吳晉賢

學程主任：

系(學程)主任、所長：

方文賢

中華民國 102 年 11 月 11 日

# 善用固態硬碟之多控制器的平行處理能力

研究生：宋鴻邑

指導教授：吳晉賢

時間：一百零二年十一月

## 中文摘要

NAND 型快閃記憶體已被廣泛應用在嵌入式系統和消費電子產品，由於它具有低功耗、快速存取、非揮發性、高抗震等特色。現今，SSD 的架構大多採用多控制器架構來處理 NAND 型快閃記憶體晶片。在傳統多控制器的結構設計 (TMCD) 下，每一個控制器都各自處理自己匯流排上的 NAND 型快閃記憶體晶片。然而，在並行多控制器的結構設計 (PMCD) 下，任何的晶片將不再侷限於任何特定的控制器，每一個閒置的控制器能透過多控制器架構的設計去存取任何的晶片。在這篇論文中，我們將提出一個新的方法，有效的利用固態硬碟多控制器的平行處理能力。當快閃記憶體轉換層 (FTL) 使用了我們所提出的方法，實驗結果證明，執行時間減少高達 5.52%。

**關鍵字：**NAND 型快閃記憶體、固態硬碟、快閃記憶體轉換層

# Abstract

NAND flash memory has been widely utilized in embedded systems and consumer electronics, because of its low-power consumption, high-performance access, non-volatility, and shock resistance. Nowadays, the architecture of SSD is using multiple controllers to handle NAND flash memory chips. Under the architecture of traditional multi-controller design (TMCD), one controller can only take responsibility for the specific NAND flash memory chips on its own bus; nevertheless, under the architecture of parallel multi-controller design (PMCD), any controllers can access any NAND flash memory chips on a SSD. In this thesis, we will propose a method to exploit multi-controller parallelism for solid-state drives regardless of TMCD or PMCD. When a flash translation layer (FTL), which provides a block device interface on top of flash memory, adopts the method, the experimental results show that the FTL for multi-controller design could reduce the total response time up to 5.52%.

**Keywords:** NAND Flash Memory, Solid-State Drives, Flash Translation Layer

## Index

Chapter 1 Introduction .....	2
Chapter 2 Background Knowledge .....	5
2.1 Related work .....	5
2.2 Solid-State Drive (SSD) .....	8
2.3 Flash Translation Layer (FTL) .....	10
Chapter 3 Problem Overview .....	14
3.1 Merge Operations of Hybrid-Mapped FTLs .....	14
3.1.1 Switch Merge .....	14
3.1.2 Partial Merge .....	15
3.1.3 Full merge .....	15
3.2 Motivation .....	17
Chapter 4 Exploiting Multi-controller Parallelism for Solid-State Drives .....	20
4.1 Overview .....	20
4.2 Handling read/write operations under TMCD and PMCD .....	22
4.3 Handling merge operations under TMCD and PMCD .....	28
4.3.1 Switch Merge .....	28
4.3.2 Partial Merge .....	29
4.3.3 Full merge .....	30
Chapter 5 Performance Evaluation .....	33
5.1 Environment setup and trace .....	33
5.2 Effect of Merge Operation .....	36
5.3 Overall performance .....	39
Chapter 6 Conclusion .....	42
References .....	44

## Index of Figures

FIG. 1: TYPES OF NAND FLASH MEMORY: SLC, MLC, AND TLC. ....	6
FIG. 2: ARCHITECTURES OF TMCD AND PMCD. ....	9
FIG. 3: THREE DIFFERENT KINDS OF MERGE OPERATIONS.....	15
FIG. 4: WORSE CASES UNDER TMCD AND PMCD. ....	18
FIG. 5: RELATIONSHIP BETWEEN DATA AND LOG BLOCKS. ....	19
FIG. 6: STRIPING TECHNIQUE: PAGE STRIPING AND BLOCK STRIPING. ....	21
FIG. 7: OVERALL ARCHITECTURE OF SSD. ....	23
FIG. 8: HANDLING READ/WRITE OPERATIONS. ....	24
FIG. 9: HANDLING REQUEST 1, REQUEST 2, REQUEST 3, AND REQUEST.....	25
FIG. 10: SWITCH MERGE UNDER MULTIPLE CONTROLLERS. ....	28
FIG. 11: PARTIAL MERGE UNDER MULTIPLE CONTROLLERS. ....	30
FIG. 12: FULL MERGE UNDER MULTIPLE CONTROLLERS. ....	32
FIG. 13: OPTIMAL REDUCED RESPONSE TIME OF MERGE OPERATIONS WITH THE .....	38
FIG. 14: ACCUMULATED RESPONSE TIME WITH AND WITHOUT THE PROPOSED METHOD	40
FIG. 15: COMPARISON BETWEEN OPTIMAL REDUCED RESPONSE TIME WITH REAL .....	41
FIG. 16: OVERALL IMPROVE RATIO. ....	41



## Index of Tables

TABLE 1: DEFAULT VALUES OF THE PARAMETERS.....	33
TABLE 2: TRACES .....	35
TABLE 3: NUMBER OF SWITCH MERGE OPERATIONS, PARTIAL MERGE .....	37





## Chapter 1 Introduction

A solid-state drive (SSD) is composed of non-volatile NAND flash memory [1]. NAND flash memory has become a popular storage unit due to the advantages of low-power consumption, high-performance access, non-volatility, and shock resistance. Compared to hard-disk drives (HDDs), SSD could provide faster read and write operation time in terms of sequential and random data access. However, there are some characteristics which are different with traditional HDDs such as out-of-place update, erase-before-write, and asymmetric operations. Therefore, FTL (Flash Translation Layer) is proposed to provide block-device emulation (e.g., hard-disk drives) for NAND flash memory by hiding the existence of out-of-place update and erase-before-write characteristics. Therefore, many existing file systems (e.g., FAT32, EXT2, and NTFS, etc) could be built on FTL without any modifications.

Currently, SSDs use a fixed architecture, called traditional multi-controller design (TMCD) [2]. Under TMCD, one controller can only access to flash memory chips on its bus due to the bus constraint which reduces execution parallelism of multiple controllers. Therefore, the performance will decrease while many I/O requests need to use those chips which belong to the same controller, and other idle controllers cannot perform the I/O requests in different buses. Because of the above

problem, the architecture of parallel multi-controller design (PMCD) [3] is proposed.

Under PMCD, any controllers can access any flash memory chips. However, if most of I/O requests are going to access a specific chip in a period of time, it could also decrease the performance.

In order to exploit the multi-controller parallelism for SSDs, striping technique [2], [17] is commonly used. It can be divided into page striping and block striping. Page striping dispatches the pages of a logical block to many buses to maximize the degree of I/O parallelism. However, page striping will increase garbage collection overhead because all invalid pages are allocated in different blocks. Block striping dispatches the pages of a logical block to the same physical block. Although the degree of I/O parallelism is limited, garbage collection overhead is relatively lower than page striping because smaller numbers of blocks are involved during garbage collection. In this thesis, we will enhance the multi-controller parallelism for SSDs by considering the collaboration of FTL and multi-controller design (i.e., TMCD and PMCD).

The rest of this thesis is organized as follows: In Chapter 2, we present background knowledge about related work, solid-state drive (SSD), flash translation

layer (FTL), and Block-based address mapping. Chapter 3 is the problem overview. Chapter 4 is to exploit multi-controller parallelism for solid-state drives. Chapter 5 introduces the trace files, and provides performance evaluation of the proposed method. Finally, Chapter 6 is the conclusion and Chapter 7 is the future work.



## Chapter 2 Background Knowledge

### 2.1 Related work

Flash memory is a non-volatile storage device that can be electrically erased and reprogrammed. Typically, flash memory can be divided into two types: NOR and NAND. NOR flash memory can be used as code storage and is feasible to execute in place for appliances with a small amount of capacity of RAM. However, NOR flash memory has some disadvantages such as low capacity and low write speed. Compared to NAND flash memory, NAND flash memory provides faster access performance and higher cell density and capacity than NOR flash memory. Therefore, NAND flash memory is suitable for data-intensive storage devices such as SSDs. In this thesis, we will focus on NAND flash memory.



As shown in Fig. 1, NAND flash memory can be divided into three types: SLC, MLC and TLC [4]. In SLC (Single-Level Cell), each cell stores only one bit of information. SLC has faster transfer speeds, lower power consumption, and higher cell endurance. Due to higher cost, it is used mostly in high-performance memory devices which need high speed and high reliability. In MLC (Multi-Level Cell), each cell stores two bits of information. Because of two bits per cell, MLC has slower transfer speeds, higher power consumption, and lower cell endurance. The advantage

of MLC is lower manufacturing cost. In TLC (Triple-level Cell), each cell stores three bits of information. A TLC-based memory device performs slower transfer speeds, higher error rates, and lower cell endurance than both SLC-based and MLC-based memory devices. The advantages of TLC are lower cost and higher density than both SLC-based and MLC-based memory devices. Furthermore, TLC chips are smaller than SLC and MLC chips for a given memory capacity. TLC is used mostly in low-end memory devices which do not focus on speed and reliability.

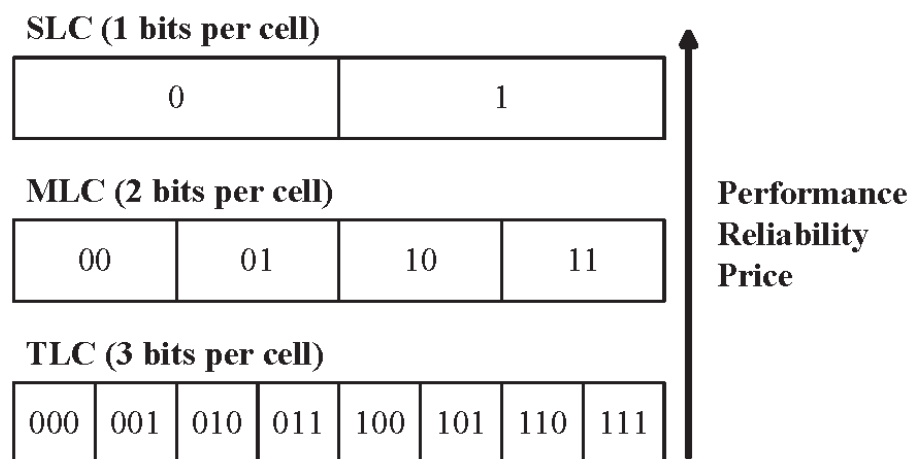


Fig. 1: Types of NAND flash memory: SLC, MLC, and TLC.

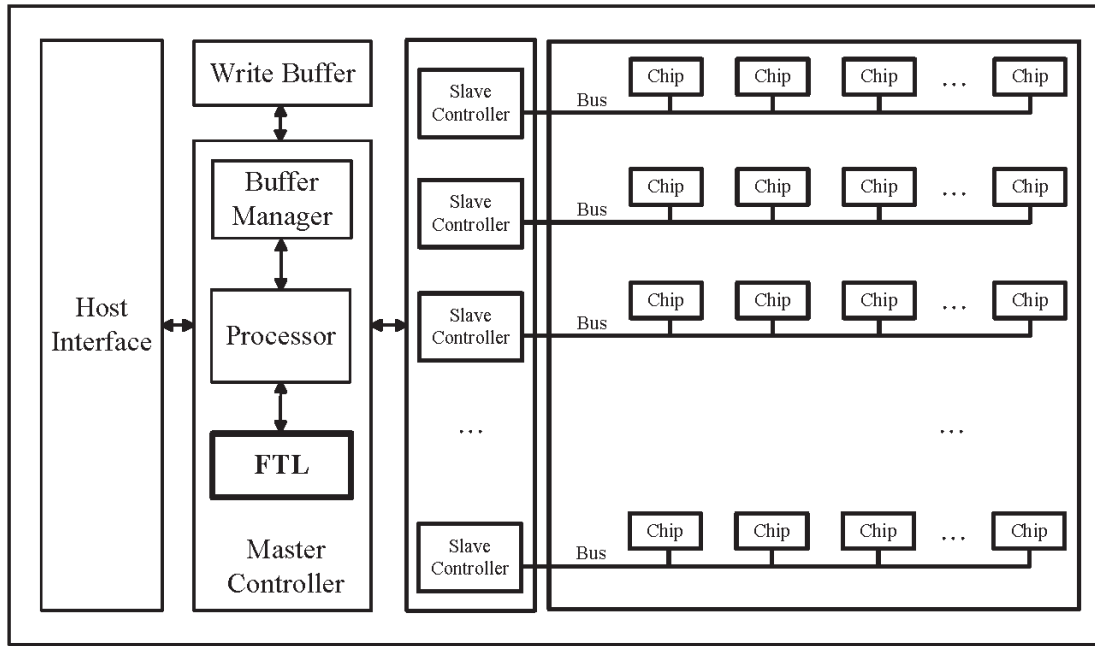
No matter which kinds of NAND flash memories, they are composed of physical blocks, and each physical block contains a set of pages (e.g., 64 pages [5]). A page is partitioned to data area and spare area. The size of the data area is 4K byte and the spare area is 128 bytes. Data will be stored in the data area, and metadata can be stored in the spare area. There are three basic operations in NAND flash memory:

read page operation, write page operation, and erase block operation. The read page operation can fetch a page from flash memory, while the write page operation can write a page to flash memory. The erase block operation can reset a block for recycling. NAND flash memory is not like hard-disk drives, because it doesn't support in-place update. Therefore, once a page is written, its residing block must be erased before the write page operation is performed on the page.

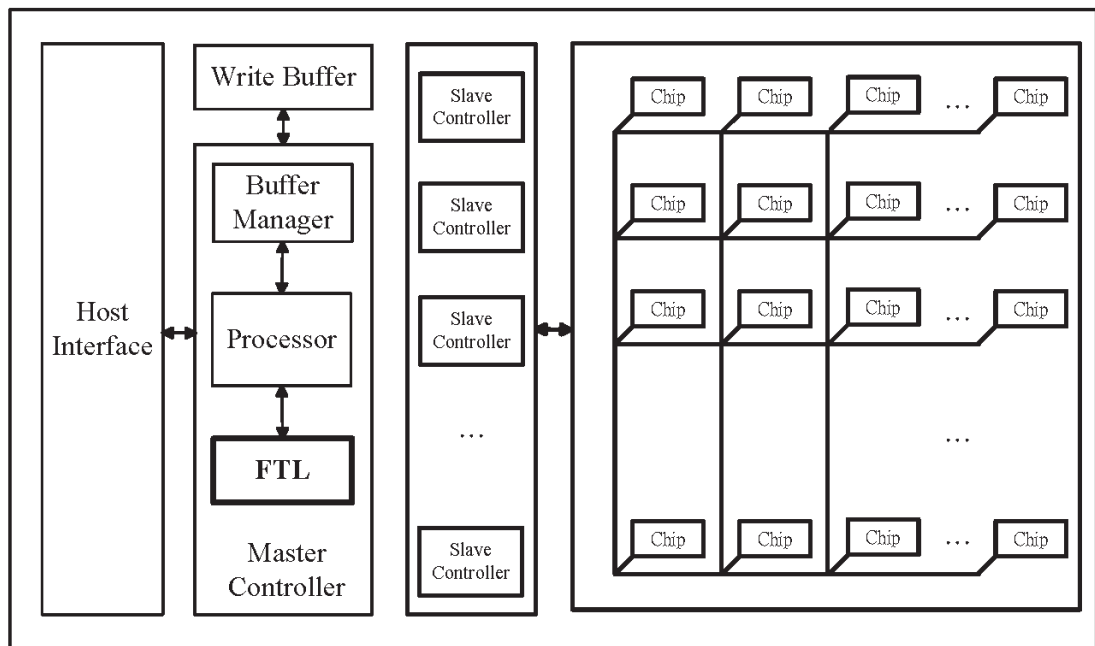


## 2.2 Solid-State Drive (SSD)

Nowadays, SSDs which consist of several NAND flash memory chips have already replaced the mechanical HDDs (Hard-Disk Drives) in terms of low-power consumption, high-performance access, non-volatility, and shock resistance. Fig. 2.(a) shows the general architecture of a SSD, called TMCD, which consists of host interface, write buffer, controllers, and flash memory chips. Generally, the controllers are classified into two categories: master controller and slave controller. Master controller is responsible for the execution of FTL, analyzing requests from the host interface, and then assigning tasks to slave controllers. Slave controllers are responsible for accessing NAND flash memory and performing basic I/O operations (e.g., read, write, and erase operations). Multiple buses are connected to the slave controllers, and the bandwidth of each bus is shared by many flash memory chips. Fig. 2.(b) shows the architecture of PMCD, any controllers in PMCD can access any flash memory chips. In the thesis, we will propose a method to exploit multi-controller parallelism for TMCD and PMCD.



(a) TMCD



(b) PMCD

Fig. 2: Architectures of TMCD and PMCD.



### 2.3 Flash Translation Layer (FTL)

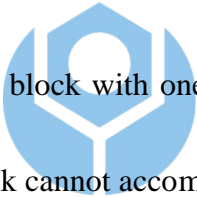
As shown in Fig. 2, FTL (Flash Translation Layer) is inside the master controller and is an interface between the sector-based file system and NAND flash memory chips. The purpose of FTL is to provide block-device emulation (e.g., hard-disk drives) for NAND flash memory by hiding the existence of out-of-place update and erase-before-write characteristics. Therefore, many existing file systems (e.g., FAT32, EXT2, and NTFS, etc) could be built on it without any modifications. There are two main functions of FTL: address translation and garbage collection. Due to the characteristic of out-of-place update, the role of address translation is to translate LBA (Logical Block Address) to PBA (Physical Block Address) by a mapping table. Due to the erase-before-write characteristic, the purpose of garbage collection is to reclaim free pages by merging and erasing blocks. According to the granularity of mapping information, FTLs can be classified into three categories: page-mapped, block-mapped, and hybrid-mapped FTLs.

Page-mapped FTLs [6] adopt a fine-grained translation method and directly translate a logical page to a physical page. In Page-mapped FTLs, a logical page can be mapped to a page in any location in NAND flash memory. For a page overwrite operation, the new page will be written to a free page, and the original page is marked

as invalid, and the mapping table will be updated accordingly. In this scheme, a logical page can be written to any physical pages and garbage collection is executed only when there are almost no free pages in the flash memory. Therefore, the cost of garbage collection is relatively small. Although this kind of mapping scheme allows for more flexible storage management, this scheme requires a large amount of RAM for the mapping table, especially when the size of flash memory increases. In order to reduce the usage of RAM, block-mapped FTLs [7], [8] adopt a coarse-grained translation method, which translates a logical page to a physical block. LPA (Logical Page Address) is divided by the number of pages in a block to get the logical block number (i.e., the quotient) and the page offset (i.e., the remainder). The logical block number is used to get the physical address of a data block, and the page offset is used to locate the target page in the data block. For a page overwrite operation, the data block is reclaimed by copying all up-to-date pages (from the data block) and the overwrite data to a free block, and then erasing the original data block. However, frequently updating in a logical block could degrade system performance and cause poor space utilization.

Combining the benefits of the page-mapped and block-mapped FTLs, hybrid-mapped FTLs [9], [10], [11], [12], [13], [14], [15], [16] have been proposed to keep the size of the mapping table small and provide reasonable performance and

high space utilization. In hybrid-mapped FTLs, most of data blocks are managed via the block-mapped scheme while the log blocks are managed via the page-mapped scheme. For a page overwrite operation, the data will be written to the log blocks. Garbage collection in hybrid-mapped FTLs is executed by reclaiming log blocks and their corresponding data blocks. When a log block needs to be reclaimed, it is merged with its corresponding data blocks. In the following, we describe two typical hybrid-mapped FTLs: BAST (Block-Associative Sector Translation) [15] and FAST (Fully-Associative Sector Translation) [16].



BAST [15] provides one data block with one dynamically allocate log block if necessary. When the target log block cannot accommodate the current write request, it is reclaimed by merging with its corresponding data block. Moreover, if all log blocks are being used, one of the allocated log blocks will be reclaimed. Furthermore, if the number of frequently updated blocks with small random writes is larger than the number of log blocks, it could cause the thrashing problem for log blocks due to frequent erasure of some log blocks with low space utilization. FAST [16] is proposed to solve the thrashing problem by using fully associative log blocks. In FAST, log blocks are divided into two groups: one log block is used for sequential writes (SW) and the other log blocks are used for random writes (RW). A SW log block

corresponds to one data block. The write request is considered as sequential writes when the SW log block is empty and the page offset of the write request is zero, or when the page offset is after that of the recently written page in the SW log block. While a sequential write cannot be satisfied by the SW log block, the SW log block is merged with its corresponding data block to get a free SW log block. A RW log block corresponds to multiple data blocks. While a random write cannot be satisfied by the RW log blocks due to no available RW log blocks, FAST selects a victim RW log block and merges the victim RW log block with its corresponding data blocks.



## Chapter 3 Problem Overview

### 3.1 Merge Operations of Hybrid-Mapped FTLs

Hybrid-mapped FTLs adopt a mixed use of page-mapped schemes and block-mapped schemes. Hybrid-mapped FTLs reserve a small number of blocks as log blocks and other blocks are used as data blocks. Data blocks store the ordinary data and are managed via block-mapped schemes. Log blocks are used to store the update data and are managed via page-mapped schemes. When a write request is going to update data to a valid page, hybrid-mapped FTLs perform the write operation to a page in the corresponding log block. If there is no available log block, a victim log block will be selected and merged with its corresponding data block(s). This is called a merge operation. When the merge operation is performed, additional cost such as page copy operations and erase operations could be invoked. Therefore, merge operations could degrade the performance due to the additional cost. Fig. 3 shows three different kinds of merge operations [14], which are as follows:

#### 3.1.1 Switch Merge

As shown in Fig. 3.(a), assume that all pages in the data block are updated sequentially, and the order of page offsets in the log block is the same as that in the data block. In this case, switch merge operation can be done by exchanging the log block with the data block and erasing the original data block. Switch merge is the most

economical merge operation because it only causes one erase operation.

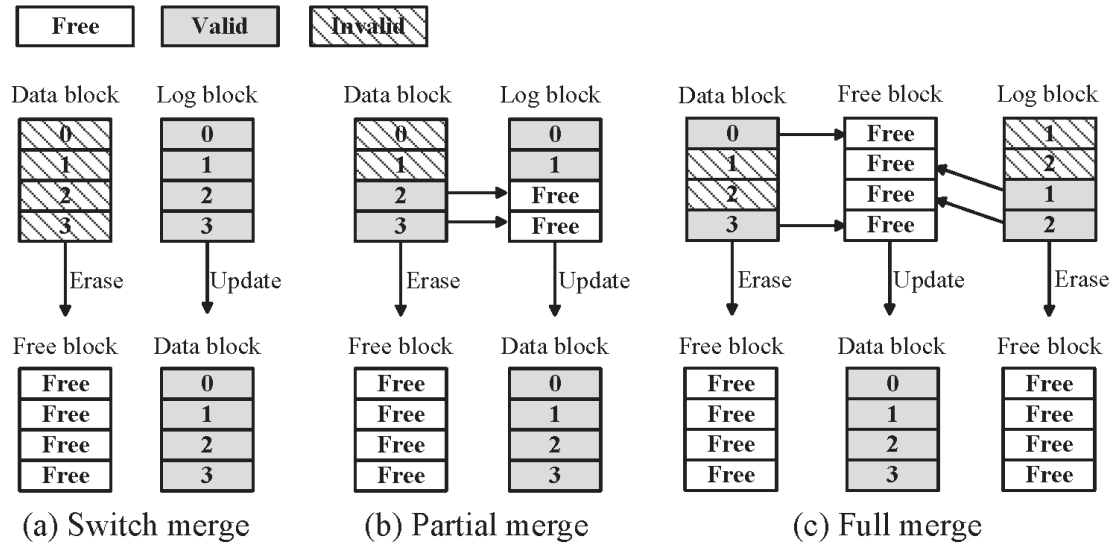


Fig. 3: Three different kinds of merge operations.



### 3.1.2 Partial Merge

As shown in Fig. 3.(b), assume that a contiguous subset of pages in the data block is updated sequentially, and the page offsets of updated pages in the log block are in a sequential and contiguous order. In this case, partial merge occurs by copying the valid pages in the data block to the free space of the log block, erasing the data block, and then assigning the log block to be the new data block.

### 3.1.3 Full merge

As shown in Fig. 3.(c), assume that a noncontiguous subset of pages in the data block is updated, and the order of page offsets in the log block and data block is

different. In this case, full merge occurs and the valid pages either from the data block or the log block are copied to a free data block. After copying the valid pages, the original data block and log block will be erased. Full merge is the least economical merge operation because it could cause many erase operations and copy operations.

According to merge operations we introduced above, we know that merge operations will influence the performance under the architecture of multi-controller design for solid-state drives.



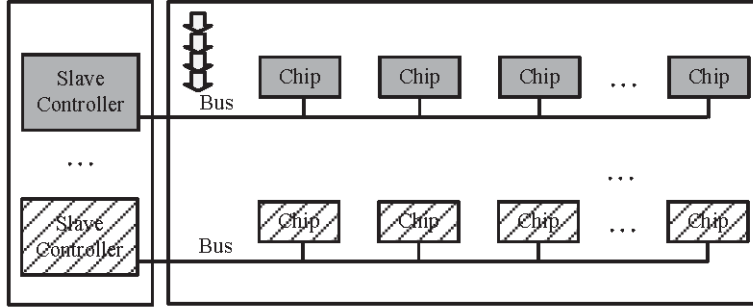
### 3.2 Motivation

SSDs consist of multiple controllers and NAND flash memory chips. SSDs have become a break-through product due to the advantages of non-volatility, lower-power consumption, faster data access, and shock-resistance. However, the multi-controller design for SSDs could cause problems at some specific situations. As shown in Fig. 4.(a), under TMCD, the first slave controller will become very busy owing to many I/O requests are going to access to its own bus in a period of time. Because of the bus constraint, other slave controllers could be idle and can't handle the I/O requests. As shown in Fig. 4.(b), under PMCD, if most of I/O requests are going to access one specific chip in a period of time, the performance will also become worse due to only one slave controller and one chip is involved and others slave controllers are idle.



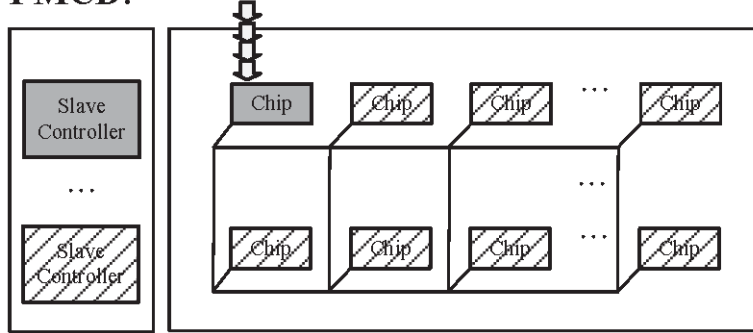


#### TMCD:



(a) Worst case under TMCD

#### PMCD:



(b) Worst case under PMCD

Fig. 4: Worse cases under TMCD and PMCD.

The read/write function of I/O requests for flash memory can be defined as follows:

*Read/Write (Start\_LPA, Size)*, which means "read/write the pages of length *Size* from the logical page address of *Start\_LPA*". As shown in Fig. 5, there are four slave controllers and four write requests. The data of these requests are sequentially written into flash memory chips. Assume that SSDs adopt a hybrid-mapped flash translation layer (i.e., data blocks and log blocks). Therefore, the first write request *Write (16,8)* will be written into a data block. Other write requests, *Write (17,3)*, *Write (21,1)*, and *Write (21,2)*, will be written to log blocks due to the out-of-place update. When I/O

requests will be written to data blocks and log blocks, the multi-controller design must consider the allocation of data blocks and log blocks. This is because the poor allocation of data blocks and log blocks could cause worse cases, as shown in Fig. 4. Furthermore, the allocation of data blocks and log blocks could also have impact on merge operations when the multi-controller design for solid-state drives is considered. For example, if a log block and its corresponding data blocks are located in the same bus or chip, TMCD and PMCD cannot exploit multi-controller parallelism to merge the log block and the data blocks. As a result, with an appropriate allocation of data blocks and log blocks, we can increase the execution parallelism of multiple controllers and reduce the response time of merge operations under TMCD and PMCD.

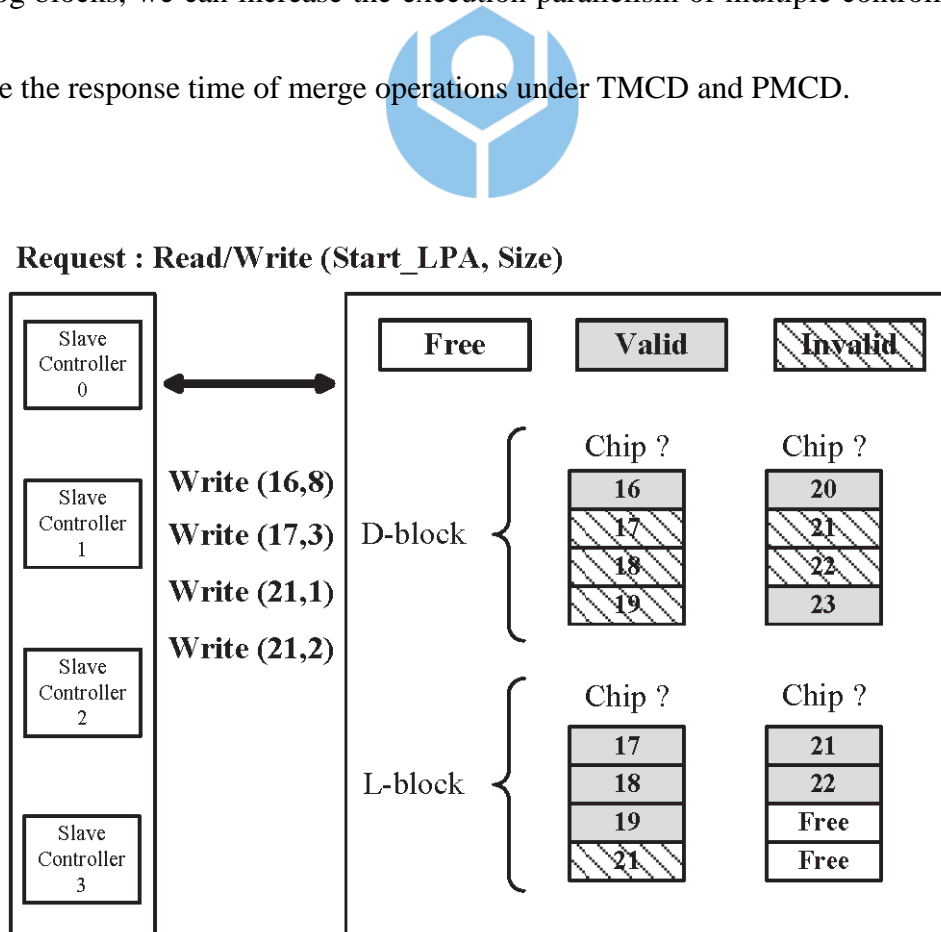


Fig. 5: Relationship between data and log blocks.

## Chapter 4 Exploiting Multi-controller Parallelism for Solid-State Drives

### 4.1 Overview

Nowadays, striping technique [2], [17] is commonly used in SSDs. It can be divided into page striping and block striping. Fig. 6.(a) shows page striping, which dispatches the pages of a logical block to many buses to maximize the degree of I/O parallelism. In Fig. 6.(a), We can see that the request *Write (0,4)* is performed by all buses. However, page striping will increase garbage collection overhead because all invalid pages are allocated in different blocks. Fig. 6.(b) shows block striping, which dispatches the pages of a logical block to the same physical block. In Fig. 6.(b), we can see that the request *Write (0,4)* is only performed by only one bus. Although the degree of I/O parallelism is limited, garbage collection overhead is relatively lower than page striping because smaller numbers of blocks are involved during garbage collection. In this thesis, we will enhance the multi-controller parallelism for SSDs by considering the collaboration of FTL and multi-controller design (i.e., TMCD and PMCD). In Chapter 4.2, we propose how to handle read/write requests under a multi-controller design of SSDs (i.e., TMCD and PMCD). Because read/write requests could be a series of accesses to data and log blocks, we propose the allocation method of data and log blocks to increase execution parallelism. In Chapter 4.3, we propose how to handle

three merge operations (i.e., switch merge, partial merge, and full merge) under a multi-controller design of SSDs (i.e., TMCD and PMCD). Because of the allocation method of data and log blocks, we can utilize the execution parallelism to improve the performance of partial and full merge operations.

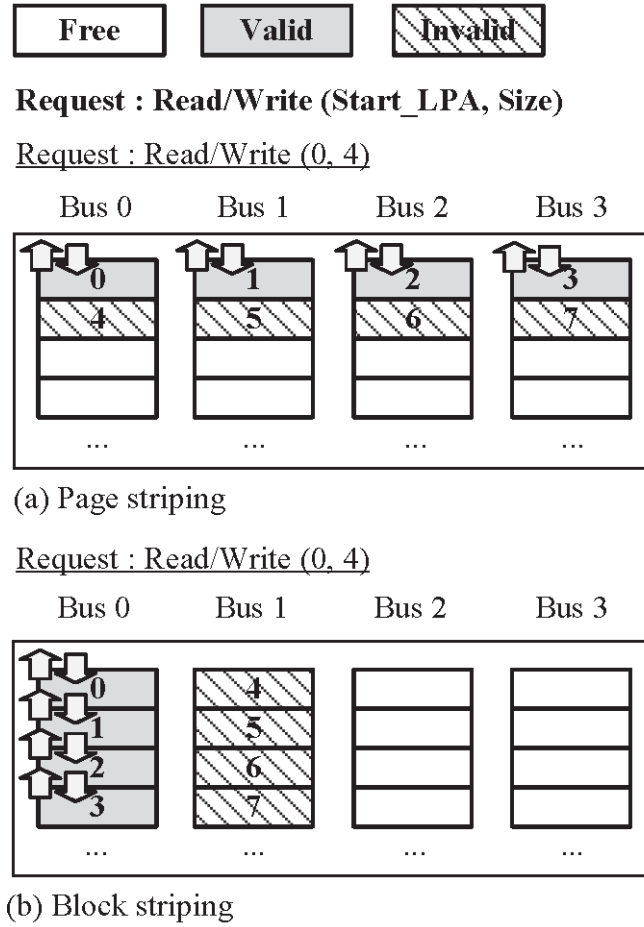


Fig. 6: Striping technique: page striping and block striping.

## 4.2 Handling read/write operations under TMCD and PMCD

In the chapter, we explain the importance of handling read/write operations under a multi-controller design of SSDs (i.e., TMCD and PMCD). As shown in Fig. 7, SSDs consist of host interface, RAM buffer, multiple controllers, and multiple flash memory chips. Each flash memory chip under a hybrid-mapped flash translation layer (i.e., FTL) can be divided into data and log blocks. As mentioned before, the read/write function for flash memory is defined as: *Read/Write (Start\_LPA, Size)*. When a file system receives a read request, FTL will handle the read request and assign controllers to read data from corresponding flash memory chips. When a file system receives a write request, FTL will allocate appropriate location (i.e., data and log blocks) in flash memory chips to write data. For example, in Fig. 7, we simplify the architecture to 4 controllers, 8 flash memory chips, and 64 pages per block. The buses of TMCD use full lines, and one controller can only access two chips which belong to the same bus. The buses of PMCD use dotted lines, and any controllers can access any chips.

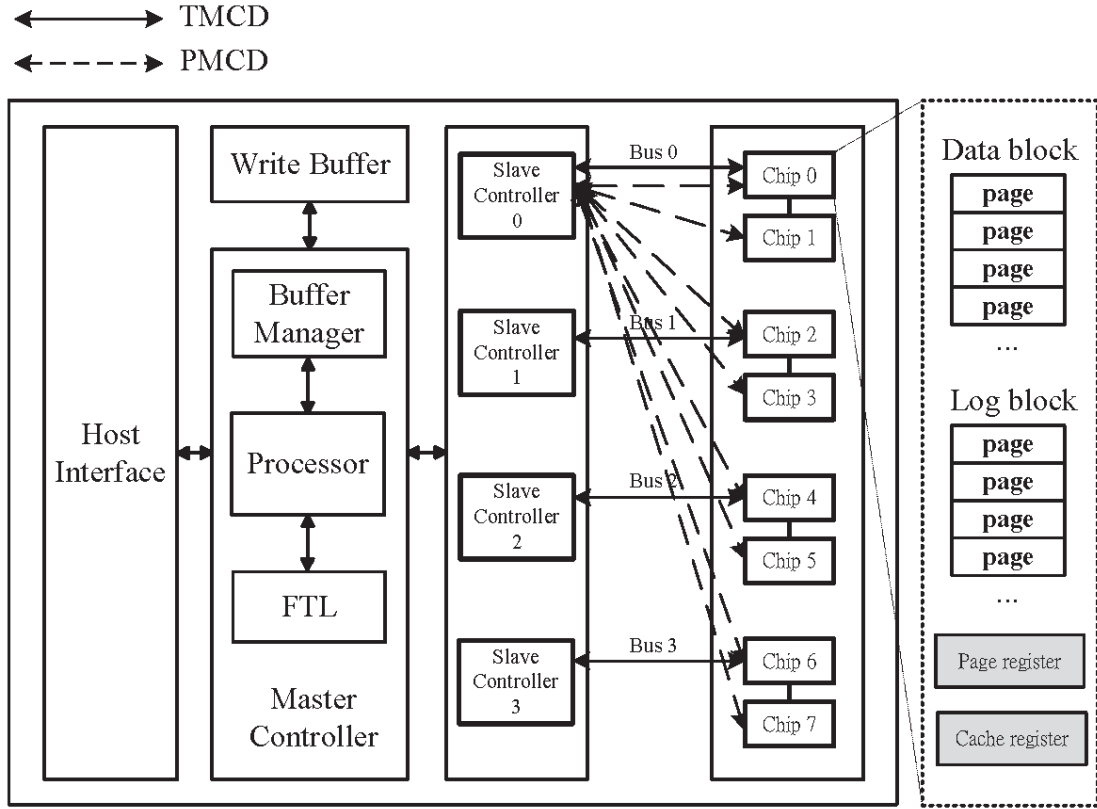


Fig. 7: Overall architecture of SSD.

As shown in Fig. 8, when a request is coming, it contains the information of *Start\_LPA* and *Size*. If it is a write request, the request will be divided into sub-requests by striping technique. Assume that the maximum size of one sub-request is one block. Sub-requests may include data blocks  $\{D1, D2, \dots, Di\}$ , or log blocks  $\{L1, L2, \dots, Lj\}$ . If it is a read request, FTL will search its mapping table and translate the read request to sub-requests that also include data blocks  $\{D1, D2, \dots, Di\}$ , or log blocks  $\{L1, L2, \dots, Lj\}$ . Then, the sub-requests will be read from the data or log blocks of corresponding flash memory chips.

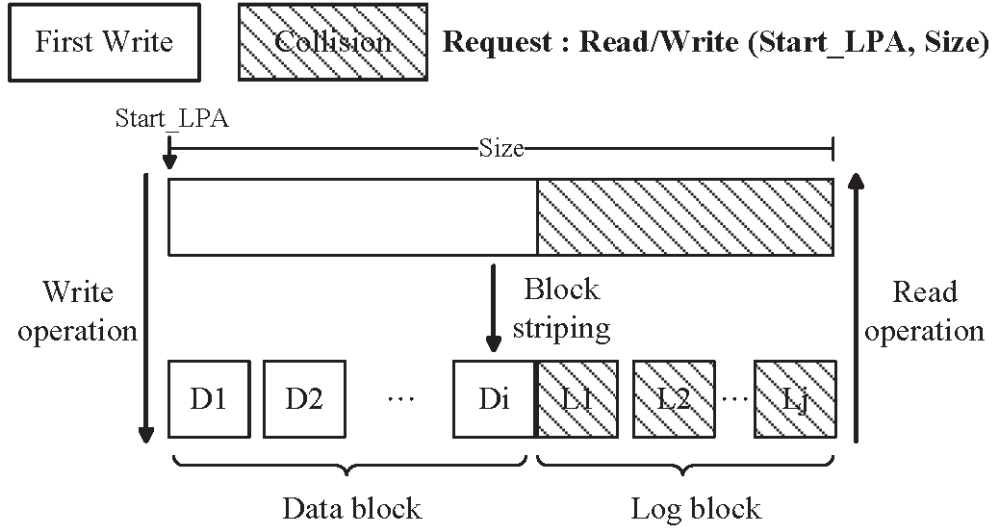


Fig. 8: Handling read/write operations.

We show how to handle the first write request. As shown in Fig. 9.(a), when *Request 1* is going to write data into data blocks, the write request is divided into sub-requests  $\{D1, D2, D3, D4\}$  by striping technique. The best case is to write the data blocks  $\{D1, D2, D3, D4\}$  into different buses under TMCD. Under PMCD, as shown in the best case of Fig. 9.(b), *Request 2* is going to write data into data blocks  $\{D1, D2, D3, D4\}$  in different chips. The worst case of Fig. 9.(a) and Fig. 9.(b) show that only one controller is involved when sub-requests are written into one bus under TMCD or one chip under PMCD. In order to effectively use multiple controllers and reduce the idle time of multiple controllers, we know that the allocation of data and log blocks for sub-requests is quite important because it can have impact on the I/O parallelism.

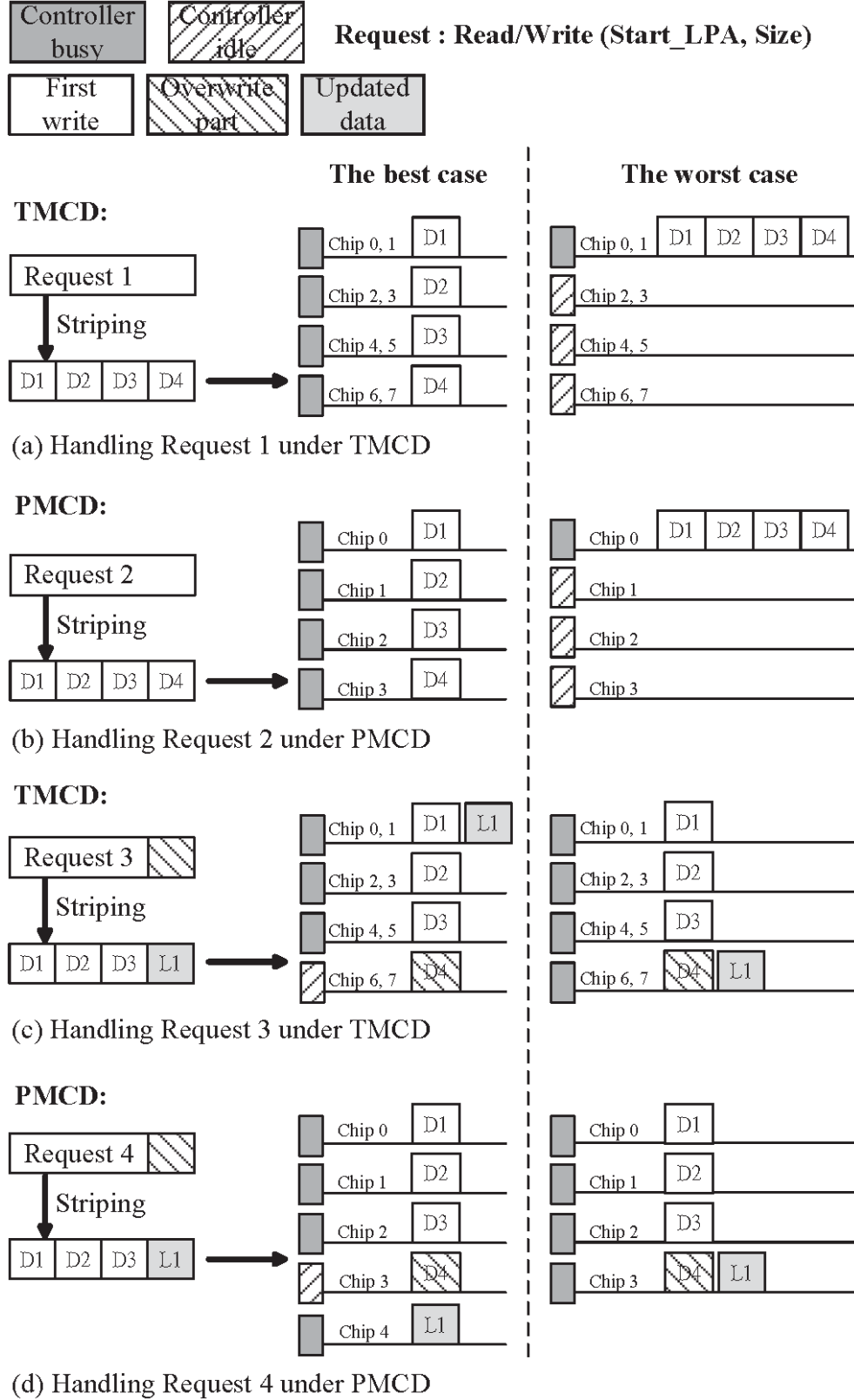


Fig. 9: Handling Request 1, Request 2, Request 3, and Request 4 under TMCD and PMCD.



We show how to handle the request with overwritten data. Due to the out-of-place update of NAND flash memory, a write request could overwrite data and cause that some sub-requests write data to data blocks  $\{D1, D2, \dots, Di\}$  or to log blocks  $\{L1, L2, \dots, Lj\}$ . As shown in Fig. 9.(c), *Request 3* will write data to data blocks  $\{D1, D2, D3\}$  and update data to a log block  $\{L1\}$ . Assume that the corresponding data block of  $\{L1\}$  is  $\{D4\}$  and partial valid data could store in  $\{L1\}$  and  $\{D4\}$ . The best case is to write the data blocks  $\{D1, D2, D3\}$  and the log block  $\{L1\}$  into different buses under TMCD. Most importantly, the log block  $\{L1\}$  should avoid locating in the same bus with the corresponding data block  $\{D4\}$ . If the log block is located in the same bus with the corresponding data block, when valid data in  $\{D4\}$  and  $\{L1\}$  is required, only one controller can be involved and reduce the I/O parallelism, as shown in the worst case of Fig. 9.(c). Similarly, under PMCD in Fig. 9.(d), *Request 4* will write data to data blocks  $\{D1, D2, D3\}$  and update data to a log block  $\{L1\}$ . The best case is to write the data blocks  $\{D1, D2, D3\}$  and the log block  $\{L1\}$  into different chips, as shown in the best case of Fig. 9.(d). The log block  $\{L1\}$  should avoid locating in the same chip with the corresponding data block  $\{D4\}$ . The worst case of Fig. 9.(d) could reduce the I/O parallelism when valid data in  $\{D4\}$  and  $\{L1\}$  is required if the log block is located in the same chip with the corresponding data block.

According to the allocation method of data and log blocks, data will be distributed to different buses under TMCD and different chips under PMCD. Thus, when read and write requests occur, we can exploit the multi-controller parallelism to access flash memory chips efficiently.



### 4.3 Handling merge operations under TMCD and PMCD

#### 4.3.1 Switch Merge

In Fig. 10.(a) and Fig. 10.(b), switch merge operation is done by switching the log block to a data block, and reclaiming the original data block to a free block. No matter where the log block is, switch merge operation only needs to erase the original data block by one controller (e.g., Controller 0). Therefore, the performance of switch merge operations with or without the proposed method is no different.

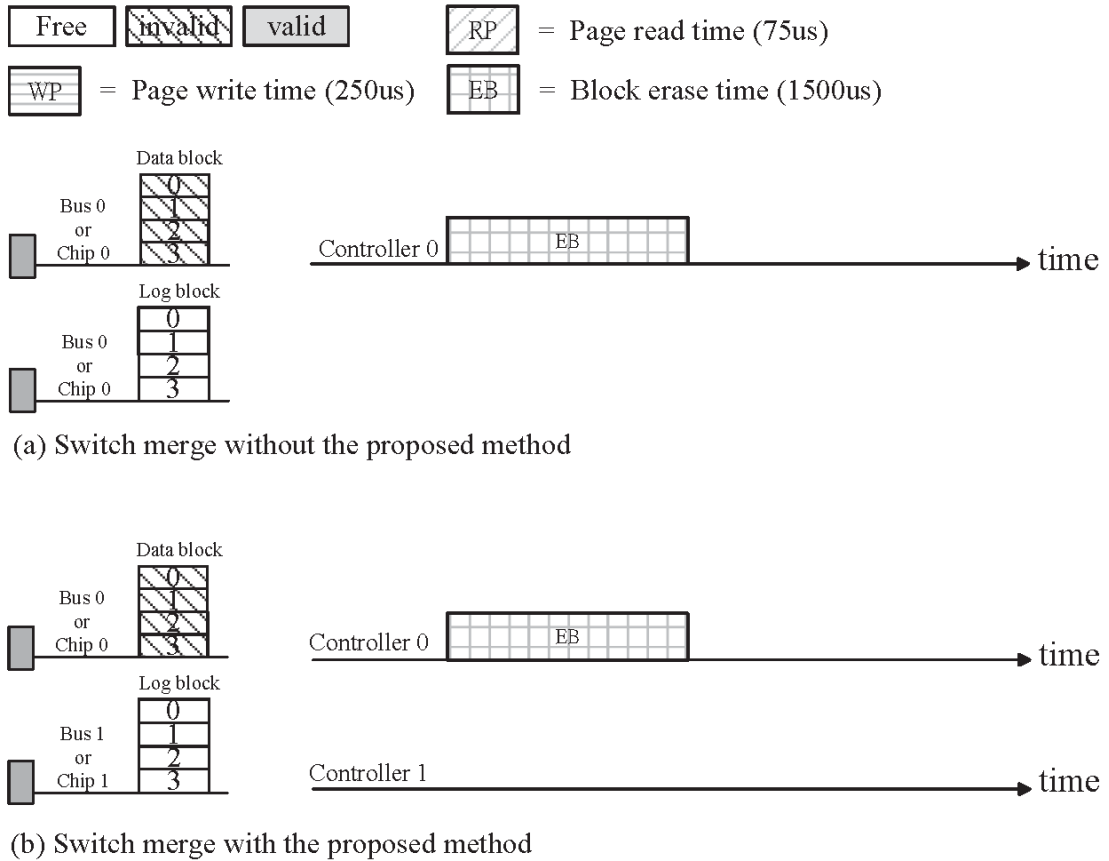
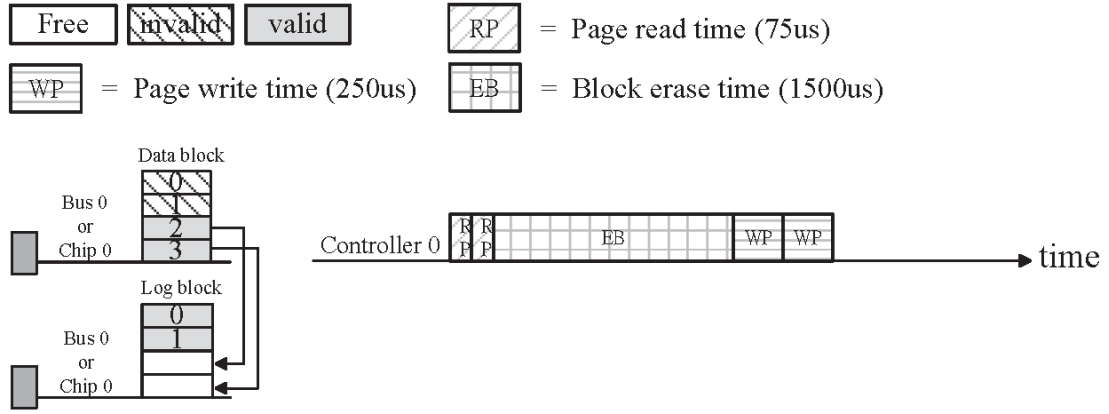


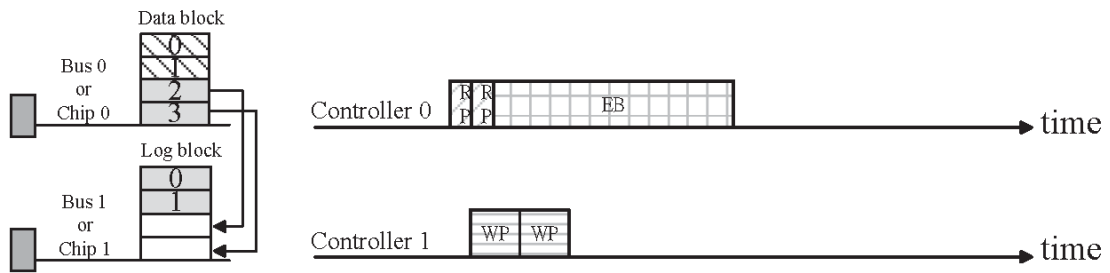
Fig. 10: Switch merge under multiple controllers.

### 4.3.2 Partial Merge

In Fig. 11.(a) and Fig. 11.(b), partial merge operation is done by copying two valid pages *Page 2, 3* from the data block to the free space of the log block, erasing the data block, and then assigning the log block to be the new data block. In Fig. 11.(a), because the data block and its corresponding log block are located in the same bus (e.g., Bus 0) under TMCD or chip (e.g., Chip 0) under PMCD, only one controller (e.g., Controller 0) can be used to read two pages, write two pages, and erase the data block. Therefore, TMCD and PMCD cannot get benefits from the allocation of the data block and the log block. However, with the proposed method in Fig. 11.(b), because the data block and its corresponding log block are located in different buses (e.g., Bus 0 and Bus 1) under TMCD or chips (e.g., Chip 0 and Chip 1) under PMCD, one controller (e.g., Controller 0) can read two pages from the data block to the buffer, and then erase the data block. At the same time, another controller (e.g., Controller 1) can write two pages from the buffer to the log block. Therefore, TMCD and PMCD can utilize the execution parallelism and improve system performance from the allocation of the data and log blocks.



(a) Partial merge without the proposed method



(b) Partial merge with the proposed method

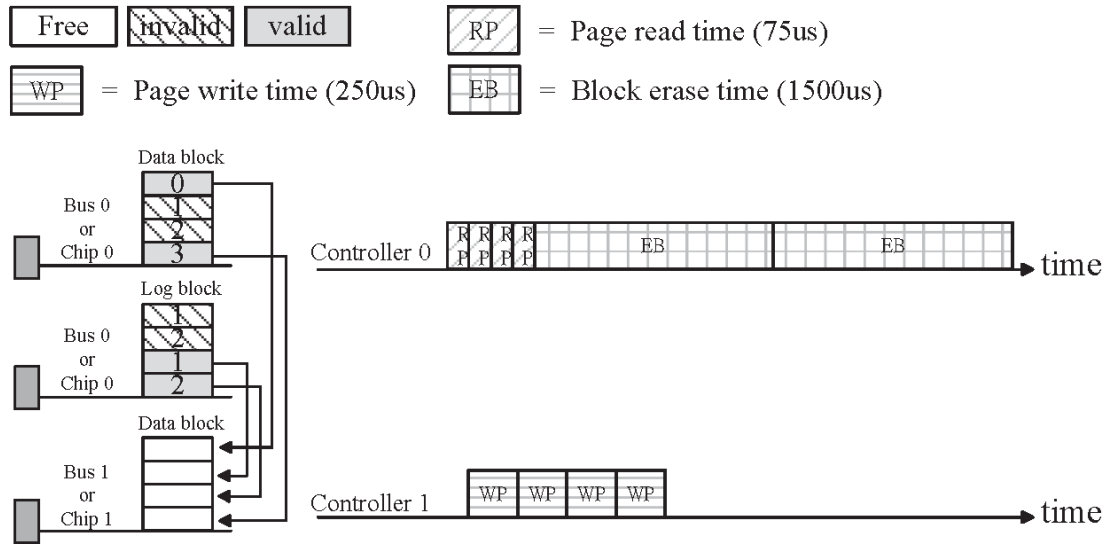
Fig. 11: Partial merge under multiple controllers.

### 4.3.3 Full merge

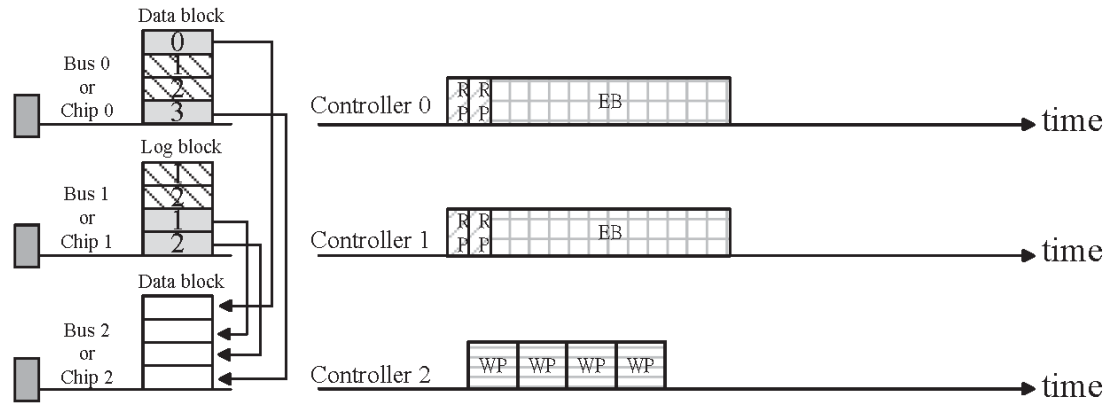
In Fig. 12.(a) and Fig. 12.(b), full merge operation is done by copying four valid pages *Page 0, 1, 2, 3* from the data block or its corresponding log block to a free data block. After copying the valid pages, the original data and log blocks will be erased.

In Fig. 12.(a), because the data block and its corresponding log block are located in the same bus (e.g., Bus 0) under TMCD or chip (e.g., Chip 0) under PMCD, one controller (e.g., Controller 0) can be used to read four pages from the data or log block

to the buffer, and then erase the data block. Another controller (e.g., Controller 1) can be used to write four pages from the buffer to the new data block. In Fig. 12.(a), although two controllers can be executed simultaneously, we still can improve the execution parallelism for TMCD and PMCD by the proposed method. With the proposed method in Fig. 12.(b), because the data block and its corresponding log block are located in different buses (e.g., Bus 0 and Bus 1) under TMCD or chips (e.g., Chip 0 and Chip 1) under PMCD, two controllers (e.g., Controller 0 and Controller 1) can read two pages from the data and log block to the buffer, and then erase the data and log blocks, respectively and simultaneously. At the same time, other controller (e.g., Controller 2) can write four pages from the buffer to the new data block. Therefore, TMCD and PMCD can utilize the execution parallelism and improve system performance from the allocation of the data and log blocks.



(a) Full merge without the proposed method



(b) Full merge with the proposed method

Fig. 12: Full merge under multiple controllers.

## Chapter 5 Performance Evaluation

### 5.1 Environment setup and trace

We have developed a trace-driven simulator to evaluate the performance of the proposed method. For comparison, we have evaluated the architecture of TMCD and PMCD, as shown in Fig. 7. In the experiments, an 80GB flash storage (i.e., 327,680 Blocks) was simulated. Assume that it contained 4 controllers, 8 flash-memory chips, and related parameters were shown in Table 1. The parameters can be obtained from the specification of SAMSUNG K9XXG08XXM NAND flash chip [5].

Table 1: Default values of the parameters

Parameters	Values	Parameters	Values
SSD capacity	80GB	Page per block	64
Number of buses	4	Page size	4KB
Chips per bus	2	Page read time	25 $\mu$ s
Dies per chip	2	Page write time	200 $\mu$ s
Planes per die	4	Data transfer time	50 $\mu$ s
Blocks per plane	5120	Block erase time	1500 $\mu$ s

As shown in Table 2, six trace files were used in the experiments. The trace files of *Financial 1* and *Financial 2* [18] were obtained by the web site of UMass Trace Repository, and were from OLTP applications running at two large financial institutions. The trace file of *MSNFS* [19] was obtained by the web site of Storage Networking Industry Association (SNIA). The original workload of *MSNFS* was collected from



MSN Storage file server for six hours. In our experiments, we used one-hour workload as the *MSNFS* trace. PCMark 7 [20] and AS SSD Benchmark [21] are popular benchmarks for computers. PCMark 7 is a complete benchmark for Windows 7. It includes 7 tests combining many individual workloads such as storage, computation, image and video manipulation, web browsing and gaming. Our *PCMark* trace was obtained from system storage suite, and the workloads of system storage suite included windows defender, importing pictures, video editing, adding music, starting applications, gaming, and windows media center. AS SSD Benchmark is a tool for measuring the speed of SSD drives. The tool contains three copy and six synthetic tests. Our *AS SSD* trace was obtained by running the three copy tests. The trace file of *WindowsPC* is one-day workload on a Windows laptop computer, which included daily activities such as program compilation, web browsing, file editing, multimedia file playing, and gaming. The last three trace files (i.e., *PCMark*, *AS SSD*, and *WindowsPC*) were collected by Process Monitor [22]. Process Monitor is similar to DiskMon [23]; however, the workloads can be collected more specific due to the function of filter which can get correct I/O operations from the corresponding process.

Table 2: Traces

Trace	Total request count	Total page access	Read ratio	Write ratio	Avg. read/write size(pages)
Financial 1	5,334,987	6,967,821	19.23 %	80.77 %	1.08 / 1.37
Financial 2	3,699,194	4,479,959	79.52 %	20.48 %	1.17 / 1.40
MSNFS	3,241,559	8,404,031	64.09 %	35.91 %	2.47 / 2.85
PCMark	387,681	6,349,836	46.49 %	53.51 %	9.97 / 37.09
AS SSD	246,957	8,925,678	33.37 %	66.63 %	69.96 / 29.10
WindowsPC	2,398,728	8,532,159	55.24 %	44.76 %	3.54 / 3.58

In Table 2, it shows the total request count, total page access, read and write ratio, and average read/write size. The *Financial 1* and *Financial 2* traces have completely different characteristics. The write ratio of *Financial 1* trace was 80.77%, but the write ratio of *Financial 2* trace was only 20.48%. In addition, the average write size of *PCMark* and *AS SSD* traces were larger than the others. It's obvious that *PCMark* and *AS SSD* could cause many switch merge operations in the experiments. In the following chapters, we first present the effect of merge operations with and without the proposed method. An overall performance with and without the proposed method under TMCD and PMCD is then presented.

## 5.2 Effect of Merge Operation

To prove that the performance can be improved by the proposed method, we have implemented the architecture of TMCD and PMCD with and without the proposed method. In the experiments, they will cause different response time, but the number of merge operations of each experiment for TMCD was equal to that of PMCD, as shown in Table 3. That was because the proposed method will not influence on the number of merge operations, but affect the total response time due to execution parallelism.

As we mentioned before, total response time of merge operations can be reduced while handling partial merge and full merge operations due to execution parallelism. As shown in Fig. 11, while handling partial merge operations, read, write, and erase operations could execute parallel. As shown in Fig. 12, while handling full merge operations, read, write, and erase operations could also execute parallel. Therefore, in Table 3, we have recorded the number of switch merge operations, partial merge operations, full merge operations, and erase count for each trace file, respectively. In Table 3, we can see that the merge operations of *Financial 1*, *Financial 2* and *WindowsPC* were mainly focused on partial merge and full merge operations. Therefore, in Fig. 13, the total response time was reduced by partial merge and full merge operations, and the optimal reduced response time of *Financial 1*, *Financial 2*

and *WindowsPC* were 321.74 seconds, 38.13 seconds and 251.43 seconds, respectively.

In Table 3, we can see the full merge operations of *MSNFS*, *PCMark*, and *AS SSD* were all zero. Therefore, in Fig. 13, the total response time was only reduced by partial merge operations, and the optimal reduced response time of *MSNFS*, *PCMark* and *AS SSD* were 267.14 seconds, 2.73 seconds, and 27.47 seconds, respectively.

Table 3: Number of switch merge operations, partial merge operations, full merge operations, and erase count for each trace file

	Switch merge	Partial merge	Full merge	Erase count
Financial 1	0 (0%)	30,190 (23.75%)	96,933 (76.25%)	1,324,306 block
Financial 2	0 (0%)	9,070 (42.18%)	12,433 (57.82%)	210,805 block
MSNFS	599 (0.06%)	1,015,837 (99.94%)	0 (0%)	1,016,436 block
PCMark	34,964 (34.07%)	67,659 (65.93%)	0 (0%)	102,623 block
AS SSD	42,539 (18.22%)	190,946 (81.78%)	0 (0%)	233,485 block
WindowsPC	21,331 (4.96%)	368,788 (85.81%)	39,638 (9.22%)	513,875 block

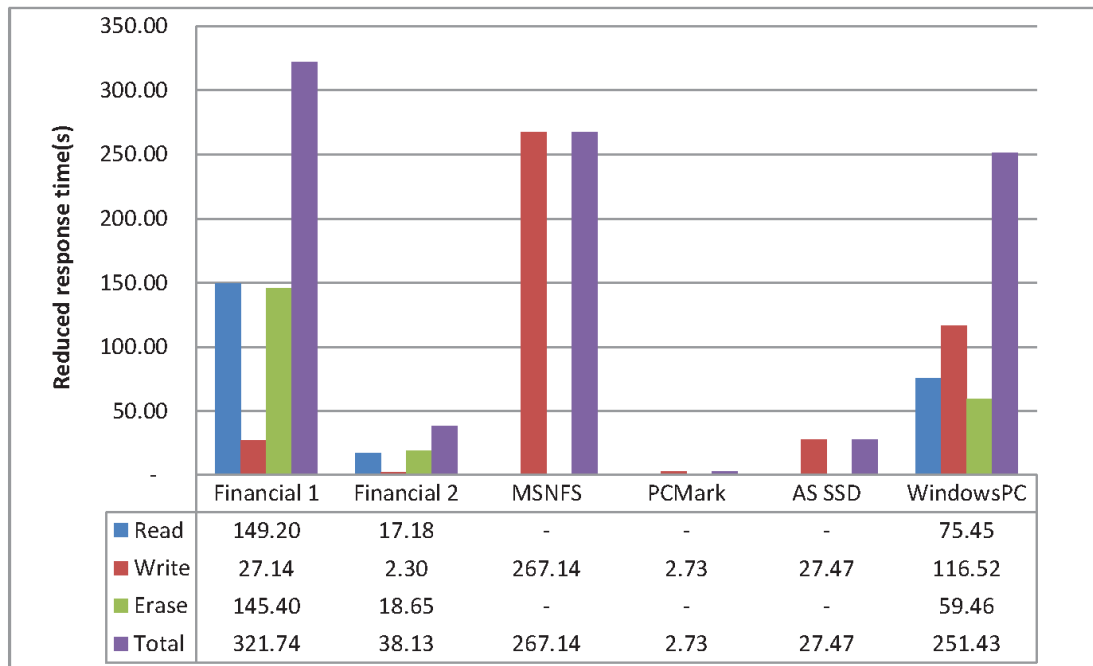


Fig. 13: Optimal reduced response time of merge operations with the proposed method.



### 5.3 Overall performance

In this chapter, we compare the overall performance of TMCD and PMCD with and without the proposed method. Fig. 14 shows the accumulated response time of TMCD without the proposed method (i.e. T-Without), TMCD with the proposed method (i.e., T-With), PMCD without the proposed method (i.e. P-Without), and PMCD with the proposed method (i.e., P-With), respectively. At first, we compare the accumulated response time of TMCD and PMCD. In Fig. 14, it is obvious that all performance of PMCD was better than TMCD. In particular, T-With and P-With were also better than T-Without and P-Without.



Second, we compare the optimal reduced time of merge operations (in Fig. 13) with the real reduced time of merge operations (in Fig. 14). Note that the reduced time of TMCD (i.e. T-Reduce) and the reduced time of PMCD (i.e. P-Reduce) were calculated from Fig. 14. For example, the optimal reduced time of the *Financial 1* trace, in Fig. 15, was 321.74 seconds, the reduced time of TMCD (i.e. T-Reduce) was 203.07 seconds, and the reduced time of PMCD (i.e. P-Reduce) was 199.68 seconds. Readers may be curious why the real reduced time of TMCD and PMCD were not equal to the optimal reduced time. That was because we assume that all controllers are idle in Fig. 11 and Fig. 12. However, in the real cases, the number of controllers is limited and the

controller may be in busy state. Therefore, there's a disparity between the real and the optimal cases. At last, we presented the ratio of performance improvement with the proposed method under TMCD (i.e., T-Improve) and PMCD (i.e., P-Improve), as shown in Fig. 16. The experimental results for the six trace files show the proposed method can reduce the total response time up to 5.52%.

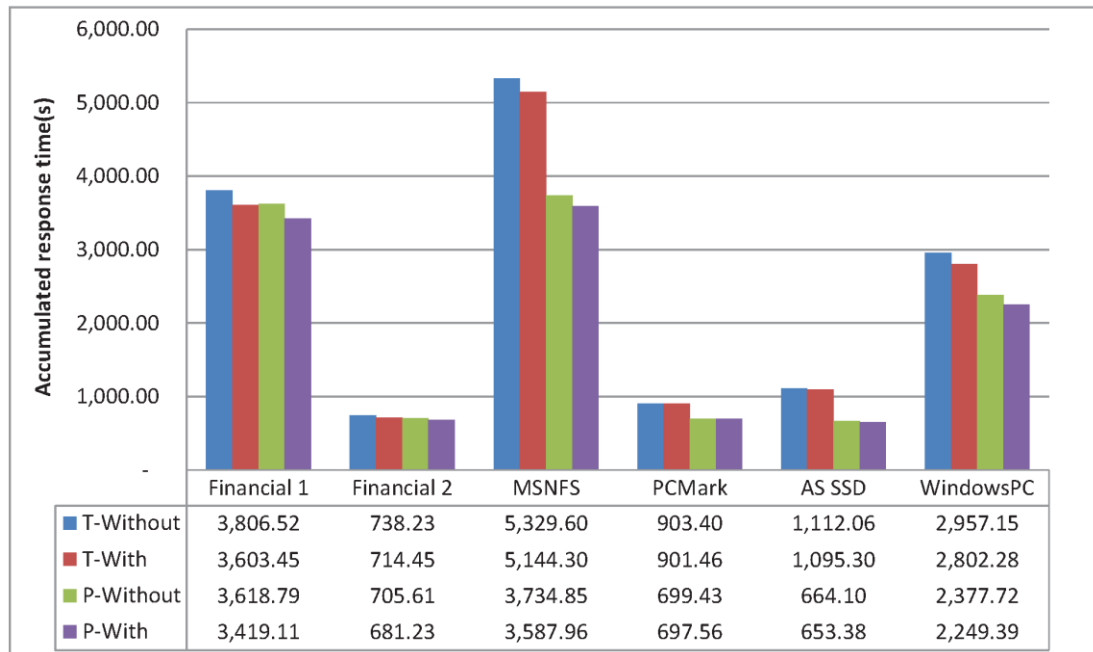


Fig. 14: Accumulated response time with and without the proposed method under TMCD and PMCD.

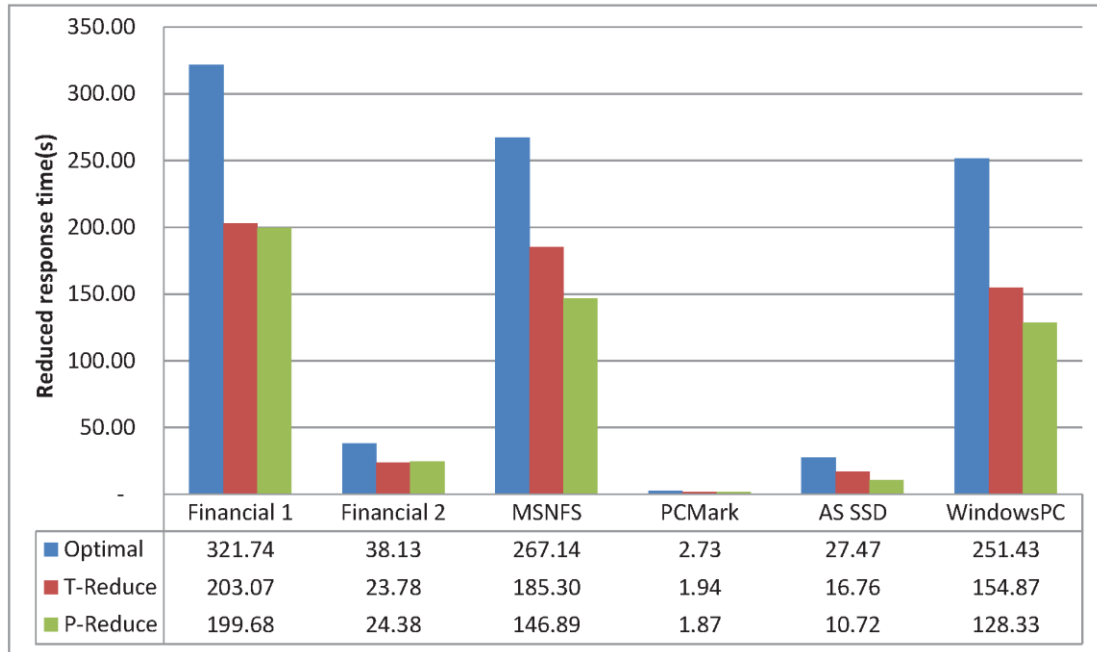


Fig. 15: Comparison between optimal reduced response time with real reduced response time.

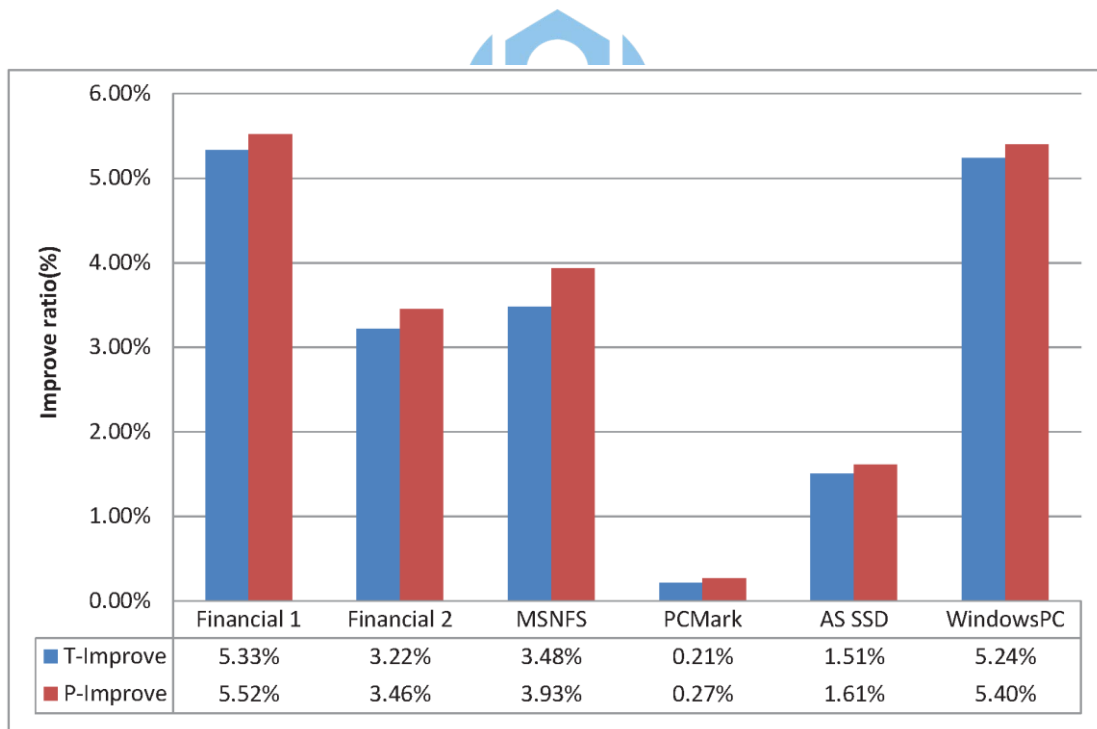


Fig. 16: Overall improve ratio.



## Chapter 6 Conclusion


Nowadays, the architecture of SSDs is using multiple controllers to handle NAND Flash memory chips. Under the architecture of traditional multi-controller design (TMCD), one controller can only take responsibility for the specific NAND flash memory chips on its own bus; nevertheless, under the architecture of parallel multi-controller design (PMCD), any controllers can access any NAND flash memory chips on a SSD. Several hybrid-mapped FTLs (i.e., the management of data and log blocks) have been also proposed to improve the overall performance of NAND flash memory. However, there are few people discuss the importance of allocation of data and log blocks under the multi-controller design of SSDs. In this thesis, we propose how to handle read/write requests under the multi-controller design of SSDs. Because read/write requests could be a series of accesses to data and log blocks, we propose the allocation method of data and log blocks to increase execution parallelism. Furthermore, we also propose how to handle three merge operations (i.e., switch merge, partial merge, and full merge) under the multi-controller design of SSDs. Because of the allocation method of data and log blocks, we can utilize the execution parallelism to improve the performance of partial and full merge operations. The experimental results from six trace files show that a hybrid-mapped FTL for multi-controller design with the proposed method can reduce the total response time

up to 5.52%.

For future research, we should further explore different access patterns and the multi-controller design. Especially, how to efficiently maximize the execution parallelism for solid-state drives will be still an important research topic. Therefore, data-intensive or computation-intensive applications should be considered when a sophisticated customization of SSD-based storage systems will be addressed.



## References

- [1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to Flash Memory", Proceedings of The IEEE, Vol. 91, No. 4, April 2003.
- [2] J.U. Kang, J.S. Kim, C. Park, H. Park and J. Lee, "A multi-channel architecture for high-performance NAND flash-based storage system", Journal of Systems Architecture: the EUROMICRO Journal, vol. 53, no. 9, p. 644-658, September 2007.
- [3] J.J. Liao and C.H. Wu, "A Multi-Controller Architecture for High-Performance Solid-State Drives", ACM SigAPP Applied Computing Review, Vol. 12, No. 4, 2012
- [4] E. Yaakobi , L. Grupp , P.H. Siegel , S. Swanson and J.K. Wolf, "Characterization and error-correcting codes for TLC flash memories", Proc. IEEE Int. Conf. Comput., Netw. Commun., pp. 486-491, 2012
- [5] SAMSUNG Electronics, "2G x 8 Bit / 4G x 8 Bit / 8G x 8 Bit NAND Flash Memory", Datasheet,  
  
<http://www.dataman.com/media/datasheet/Samsung/K9WBG08U1M K9KAG08U0M K9NCG08U5M rev10.pdf>, 2007.
- [6] Intel Corporation, "Understanding the Flash Translation Layer (FTL) Specification", ApplicationNote AP-684, Dec 1998.
- [7] A. Ban, "Flash File System", US Patent No. 5,404,485, 1995.
- [8] A. Ban and R. Hasharon, "Flash File System Optimized for Page-Mode Flash Technologies", US Patent No. 5,937,425, 1999.
- [9] S. Lee, D. Shin, Y.J. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems", ACM SIGOPS Operating Systems Rev., vol. 42, no. 6, pp. 36-42, Oct 2008.
- [10] S. Kang, S. Park, H. Jung, H. Shim and J. Cha, "Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices", IEEE Transactions on Computers, vol. 58, no. 6, pp. 744-758, JUNE 2009
- [11] C.H. Wu, H.H. Lin, and T.W. Kuo, "An Adaptive Flash Translation Layer for

High-Performance Storage Systems”, IEEETrans. Computer-Aided Design of Integrated Circuits and Systems, vol. 29, no. 6, pp. 953-965, June 2010.

[12] M.L. Chiao and D.W. Chang, “ROSE: A Novel Flash Translation Layer for NAND Flash Memory Based on Hybrid Address Translation”, IEEE Transactions on Computers, vol. 60, pp. 753-766, 2011.

[13] S. Bai and X.L. Liao, “A Parallel Flash Translation Layer Based on Page Group-Block Hybrid-Mapping Method”, IEEETrans. Consumer Electronicss, vol. 58, pp. 441-449, 2012.

[14] J.U. Kang, H. Jo, J.S. Kim, and J. Lee, “A Superblock-Based Flash Translation Layer for NAND Flash Memory”, Proc. Sixth ACM and IEEE Intl Conf. Embedded Software, pp. 161-170, 2006.

[15] J. Kim, J.M. Kim, S.H. Noh, S.L. Min and Y. Cho, “A Space-Efficient Flash Translation Layer for Compact Flash Systems”, IEEETrans. Consumer Electronics, vol. 48, no. 2, pp. 366-375, May 2002.

[16] S.W. Lee, D.J. Park, T.S. Chung, D.H. Lee, S. Park and H.J. Song, “A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation”, ACM Trans. Embedded Computing Systems, vol. 6, no. 3, July 2007.

[17] S.K. Park, Y. Park, G. Shim and K.H. Park, “CAVE: channel-aware buffer management scheme for solid state disk”, Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 346-353, May 2011.

[18] K. Bates and B. McNutt, OLTP I/O Trace,  
<http://traces.cs.umass.edu/index.php/Storage/Storage>, 2007.

[19] V. Sharda, S. Kavalanekar and B. Worthington, Block I/O Traces,  
<http://iota.snia.org/traces/158>, 2008.

[20] Futuremark Corporation., PCMark 7,  
<http://www.futuremark.com/benchmarks/pcmark7>.

[21] A. Schepeljanski, AS SSD Benchmark,  
<http://alex-is.de/PHP/fusion/downloads.php?cat id=4&download id=9>.

[22] Process Monitor, <http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx>

[23] DiskMon, <http://www.sysinternals.com/utilities/diskmon.html>.



# 國立臺灣科技大學博碩士論文授權書

(本授權書裝訂於紙本論文內)

本授權書所授權之論文為宋鴻邑 (M10002137) 在國立臺灣科技大學電子工程系 102 學年度第 1 學期取得碩士學位之論文。

論文題目： 善用固態硬碟之多控制器的平行處理能力  
指導教授： 吳晉賢

茲同意將授權人擁有著作權之上列論文全文(含摘要)，依下述授權範圍，以非專屬、無償授權本校圖書館及國家圖書館，不限地域、時間與次數，以紙本、微縮、光碟或其他數位化方式將上列論文重製典藏，並提供讀者基於個人非營利性質之線上檢索書目、館內閱覽、或複印。

授權人

宋鴻邑

吳晉賢

簽章

(請親筆正楷簽名)

宋鴻邑

吳晉賢

備註：

1. 授權人不因本授權而喪失上述著作之著作權。
2. 本授權書請授權人簽章後，裝訂於紙本論文內。

中 華 民 國

102 年 11 月 11 日