Design and Analysis of Flash Translation Layers for Multi-Channel NAND Flash-based Storage Devices

Sang-Hoon Park, Seung-Hwan Ha, Kwanhu Bang, *Student Member*, IEEE, and Eui-Young Chung, *Member*, IEEE

Abstract — *NAND Flash-based Storage Devices (NFSDs) have been replacing the conventional magnetic storage devices in many consumer electronic systems. One of the advantages of NFSDs is their read/write bandwidth, which is higher than that of the magnetic storage devices. For further increase of their bandwidth, high-end NFSDs employ multi-channel and multi-way architectures in which it is possible to access the NAND Flash Memories (NFMs) in parallel for amortizing the long latency of NFMs. Even though this architecture provides higher bandwidth from the hardware perspective, the overall performance of an NFSD critically depends on how efficiently the multiple channels and ways are utilized. In this regard, the key design component is an intermediate software layer called Flash Translation Layer (FTL), since it manages the hardware resources as well as data. To the best of authors' knowledge, this is the first work to propose a general method to design an FTL for multi-channel / multi-way NFSDs (FTL-MM). The proposed design method consists of two steps. First, we design an FTL for a single-channel / single-way NFSD (FTL-SS). Second, we extend the FTL to support an NFSD with an arbitrary number of channels and ways. To prove the generality and effectiveness of the proposed method, we apply the method to three well-known FTLs. The experimental results indicate that the FTLs enhanced by our approach are comparable to the ideal FTL and that their performance is scalable to various channel / way architectures. Quantitatively speaking, the average channel utilization decreases by at most 10%, when we increase the number of channels and ways up to four*[1].

Index Terms — NAND Flash Memory (NFM), storage, Multi-channel, Multi-way, Flash Translation Layer (FTL)

I. INTRODUCTION

Magnetic storage devices such as Hard Disk Drives (HDDs) have been the *de-facto* standard mass-storage devices in most electronic devices. However, recently introduced NAND Flash-based Storage Devices (NFSDs) have been gradually replacing HDDs, since they are superior to HDDs in several aspects – low power, small form factor, and shock-resistance. Furthermore, they enlarge the application area of storage devices thanks to the small form factor. NFSDs are now employed in many hand-held devices such as cellular phones, Mobile Internet Devices, digital cameras in addition to the traditional applications such as Personal Computers and digital TVs. More importantly, NFSDs provide a higher Input / Output (IO) bandwidth than conventional magnetic storage devices. This is an unbeatable merit of NFSDs in data-intensive applications, since the CPU-IO performance gap has become larger. More precisely, the CPU performance improvement ratio in the last decade is roughly 20 times larger than the HDD performance improvement ratio for the same period.

The key challenge in improving the performance of NFSDs is how to mitigate the long latency of NAND Flash Memories (NFMs). This problem has become even severer due to the advent of Multi-Level Cell (MLC) NFMs that need longer data access time than the conventional NFMs called Single-Level Cell (SLC) NFMs. The merit of MLC NFMs over the SLC NFMs is bit cost. Their bit cost is much lower than that of SLC NFMs, since they store multiple bits in a cell while SLC NFMs store a bit per cell.

Typically, NFSDs consist of two parts – NFMs and a controller. The controller is in charge of managing the incoming data and the hardware resources including NFMs. For this reason, the overall performance of an NFSD critically depends on not only its architecture, but also its management policy. The management policy is typically implemented as an intermediate software layer called Flash Translation Layer (FTL). From the data management perspective, one of its important roles is to translate the logical addresses of the incoming data to physical addresses. This translation requires non-trivial computational efforts due to the limited lifetime of NFM cells, erase-before-write and other complications.

The architecture of an NFSD controller affects the computation of FTLs. For instance, contemporary NFSD controllers employ a multi-channel and multi-way architecture for increasing read/write bandwidth. In this architecture, multiple communication paths (*i.e.,* channels and ways) are provided between the controller and NFMs. Even though such architecture provides more bandwidth from the hardware perspective, the effective bandwidth of an NFSD critically depends on how efficiently the increased hardware resources are utilized by an FTL. Although many previous works studied the performance improvement of FTLs, no published work has addressed the performance issue of FTLs for multi-channel and multi-way NFSDs. That is, the performance issue has been studied only in single-channel and single way NFSDs.

[1] This work was supported in part by the IT R&D program of MKE/IITA 2009-S005-01(Development of Configurable Devices & S/W environment), by the Korea Research Foundation Grant funded by the Korean Government (MEST) (KRF-2007-313-D00578), and by Hynix Semiconductor Inc..

Sang-Hoon Park, Seung-Hwan Ha, Kwanhu Bang, and Eui-Young Chung are with School of Electrical and Electronic Engineering, Yonsei University, Seoul, 120-740, Korea (e-mail: {soskhong, shha}@dtl.yonsei.ac.kr, {lamar49, eychung}@yonsei.ac.kr).

Contributed Paper
Manuscript received July 15, 2009

In this work, we focus exactly on the shortcomings of previous FTL works with the following contributions. First, we propose a general design method of FTLs targeting multi-channel and multi-way NFSDs. Second, we quantitatively prove the effectiveness of our method by applying it to three well-known FTLs to make them support multi-channel and multi-way NFSDs. Finally, we define the performance upper bound of FTLs by introducing the *oracle FTL* that is only possible by the offline analysis of traces.

The rest of this paper is organized as follows. In Section II, we summarize the basics of NFSDs and the related work. In Section III, we describe the proposed method in detail. Then, we show its effectiveness through the extensive experimental results in Section IV followed by a conclusion in Section V.

II. Preliminaries And Related Work

A. NFSD Architecture

A typical NFSD architecture is shown in Fig. 1. An NFSD consists of two parts – an NFSD controller and NFM chips.

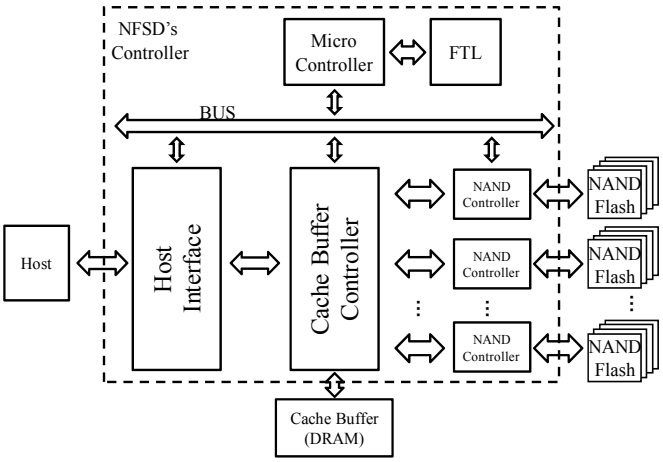


Fig. 1. A typical NFSD architecture.

The controller and NFM chips communicate with each other through a shared bus called channel. Due to the long latency of NFM chips, the channel often becomes the performance bottleneck of NFSDs. There are two popular ways to improve the channel performance. One of them is to connect multiple NFM chips to a channel such that the channel accesses them in a time-multiplexed manner to hide their long latencies. We call this technique *n-way interleaving* if n NFM chips are connected to a single channel. The other technique called *channel striping* is to increase the number of channels for accessing NFM chips in parallel. In other words, the channel bandwidth is proportional to the number of channels.

Example 1: Fig. 2 shows how a multi-channel / multi-way NFSD architecture efficiently manages the write operations, using a 2-channel / 4-way NFSD architecture. A write operation consists of three phases – Cmd, Data, and Program. The write command and the data to be written are asserted to an NFM in Cmd phase and in Data phase, respectively. In the Program phase, the asserted data is written to the NFM memory cells. The time required for the Program phase is much longer than the other two phases and the interleaving technique effectively hides the long latency of this phase by overlapping multiple write operations within a single channel. In Channel 0, we only need three ways, since the latency of three overlapped write operations is long enough to hide the latency of the Program phase, hence increasing the number of ways larger than 3 is of no use from the performance perspective. Further parallelism exploitation is possible by having more channels, since the interleaved write operations are also applicable to the additional channels as shown in Channel 1. Similarly, read operations are also benefited in this architecture.

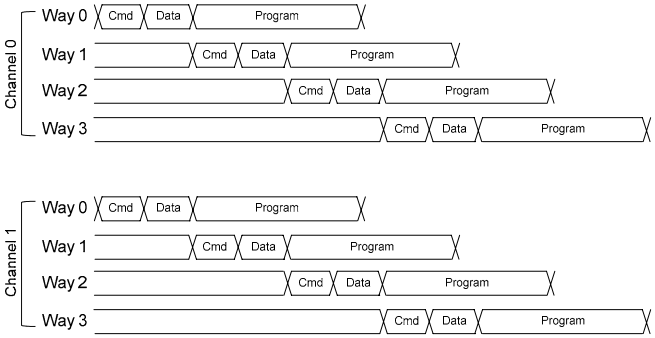


Fig. 2. Concurrent interleaved write operations in 2-channel / 4-way NFSD architecture.

The NFSD controller includes several sub-components. The host interface logic is in charge of communicating with the host machine. Also, many higher-performance NFSDs employ DRAMs as caches for increasing the performance by exploiting the data locality principle. A cache buffer controller inside the NFSD controller manages the cache buffer. The last component inside the NFSD controller is a micro-controller on which an intermediate software layer called *Flash Translation Layer* (FTL) is running. The FTL manages the overall hardware resources and incoming data, hence its quality critically affects on the overall performance of NFSDs. Note that the channels and ways are also managed by the FTL. We will discuss more details of FTLs in Section II.C.

B. NFM chip

An NFM chip has large memory cell arrays that store the incoming data. The memory cells in an NFM chip are hierarchically managed. More specifically, each memory array consists of fixed-size blocks and a block is partitioned into fixed-size pages. A page is also partitioned into fixed-size sectors. A block is the basic unit of erase operations, while a page is the basic unit of read and write operations.

Even though NFM chips are advantageous over the conventional non-volatile memories and magnetic storages, they still have a few weak points. First, they have an erase-before-write property, meaning that a block erase must be preceded to update even a single page inside the block.

Moreover, the time for a block erase operation is several times longer than write / read operations. Hence, frequent block erase operations will severely decrease the overall performance. Second, a block of NFMs has its limited lifetime that is often measured by the maximum erase count. For the NFM reliability, it is very important to write data to blocks as uniformly as possible. Finally, the read / write performance of NFM chips is asymmetric. For instance, the write performance is less than 30% of the read performance in [2]. For this reason, the write performance improvement is a challenging issue in NFSD design.

C. Flash Translation Layer

Flash Translation Layer (FTL) is an intermediate software layer between the host file system and NFMs. Its major role is translating the logical addresses given by the host system into the physical addresses of NFMs. In addition, FTLs perform some extra tasks such as wear-leveling and garbage collection. These tasks are mainly required to hide the weak points of NFMs mentioned in Section II.B.

Wear-leveling is a management policy to prolong the lifetime of flash cells by evenly distributing the number of erasing to each block. For this purpose, it often requires the data movement among flash blocks, and then the flash blocks only with invalid pages are erased and reserved for future use. We call this procedure *garbage collection*. Since these tasks may incur inordinate extra erase operations, the performance of NFSDs critically depends on the quality of these tasks. Hence, the reduction of erase operations has been the main objective of previous works.

One intuitive address-mapping scheme is to translate the logical addresses into the physical addresses in page-level, since every read / write data is processed in the unit of page in NFMs. This idea was employed in the FTL called *page-mapped FTL* [11], [12]. This scheme translates a logical page address into the address of any unused physical page and the mapping information is stored in an address-mapping table. A drawback of this scheme comes from the size of the address-mapping table, since its size is proportional to the number of pages. For this reason, it is rarely used in large capacity NFSDs.

To mitigate this issue, FTLs with two-level address translation schemes were proposed. We call them *block-mapped FTLs*. Simply speaking, the block-mapped FTLs perform the address translation in two steps. In the first step, a logical page is mapped to one of any unused physical blocks, and the mapping information is recorded into the address-mapping table. Note that the size of this address-mapping table is much smaller than that of page-mapped FTLs, since a block usually consists of 64 or 128 pages in conventional NFMs. In the second step, a physical page inside the chosen block is allocated to the logical page. Also, most block-mapped FTLs employ the concept of temporary blocks for further accelerating the mapping table search especially for write operations. This concept was first introduced in [3], where the authors classified the blocks into two categories – data blocks and log blocks. The log blocks are temporary storage blocks and the number of log blocks is much smaller than that of the data blocks. In a write operation, data are always written to the log block, hence the mapping table search time is marginal due to the small number of log blocks. However, after the log blocks are fully used up, the FTLs have to move the data in log blocks to data blocks for future write operations. These extra operations can severely degrade the overall system performance and more recent block-mapped FTLs this attempted to reduce these extra operations.

The aforementioned FTLs mostly focused on the address-mapping granularity only for a specific architecture, *i.e.* single-channel / single-way architecture. They assumed that all blocks (pages) have identical physical characteristics in block-mapped FTLs (page-mapped FTLs). However, block-mapped (page-mapped) FTLs need to distinguish the pages (or blocks) depending on their physical locations in the multi-channel / multi-way architecture, since their physical locations affect on the overall performance. More specifically, the efficiency of channel striping and interleaving critically depends on the choice of blocks that determines the parallel accessibility of NFMs. This is exactly what we tackle in this paper, and we will present the details of our method in Section III.

D. Related Work

Most previous works focused on the single-channel / single-way architecture. There are only a few works on multi-channel / multi-way architectures. The work in [1] analyzed the impact of multiple channels and multiple ways on the performance of NFSDs. A similar work can be found in [8]. The authors used a 2-channel / 4-way NFSD architecture, and its performance was about 1.7 times higher than the compared HDD. The authors in [6] proposed an improved channel striping technique for increasing the channel utilization by arranging the channel access order adaptive to the size of incoming data. Even though these techniques improve the overall system performance, they did not analyze the computing overhead incurred to the FTLs. More importantly, none of these works showed that the performance gain achieved by their methods is scalable with respect to the number of channels and ways. The scalability is a very important metric to measure the general applicability of a technique to an arbitrary number of channel / way architectures. If a technique is not scalable, it cannot be used for higher performance NFSDs which employ more channels and ways.

III. FTL FOR MULTI-CHANNEL / MULTI-WAY NFSDS

A. Overview

The proposed FTL design method assumes that there is already an FTL designed for single-channel / single-way NFSDs. The objective of the proposed method is to extend the

FTL for single-channel / single-way (FTL-SS) to support multi-channel / multi-way NFSDs (FTL-MM), while maintaining the efficiency of the extended FTL greater than or equal to that of FTL-SS. For this reason, we do not focus on designing an FTL-SS in this work. However, we show the effectiveness and generality of our method by applying it to several well-known FTLs. For this reason, an FTL extended by our method inherits the nature of its single-channel / single-way version. We briefly summarize these FTLs in Section III.B. Then, we address the critical design parameters of our method in Section III.C. In Sections III.D and III.E, we describe the address translation scheme in our method and the address-mapping structure we propose, respectively.

B. FTLs for single-channel / single-way NFSDs

Many research groups have performed the research on FTL-SSs. Most of these FTLs aimed at reducing the count of extra operations (*i.e.,* valid page copying and erase operations), since they can severely degrade the overall performance of NFSDs. The occurrence of extra operations is mainly due to the wear-leveling and garbage collection.

Among many previous FTLs, we will summarize three well-known FTLs – BAST [3], FAST [4], and Superblock FTL [5]. We also utilize them in two aspects. First, we extract the common features of these three FTLs. Based on the features extracted, we build a general method for extending FTL-SSs. Second, we utilize them to validate the generality and effectiveness of our method through extensive experiments.

Simply speaking, the previous FTLs mostly employed a block-mapped address translation scheme, hence being conceptually similar to each other. Their major difference lies in the associativity between the data blocks and log blocks, which critically affects the number of the extra operations.

In the following, we summarize the aforementioned three FTLs from the associativity perspective.

- BAST (Block-Associative Sector Translation) [3] is a log block based block-mapped FTL. When a write operation is asserted with data and its corresponding logical address, BAST first searches a block from an address-mapping table. Then, it allocates a page based on the page offset which corresponds to the lower bits of the logical address. There are three types of blocks in BAST – free block, data block, and log block. A free block is a clean storage and is reserved for future data writes. A data block allocates a page for storing the data being written. We call the allocated page *valid page*. If the logical address of a write operation is identical to that of one of previous writes, a *valid page* will be allocated for the new write operation. In this case, BAST *invalidates* the page and writes the new data to a log block. We call this procedure *an update*. BAST allocates a single log block for each data block, whenever an *update* occurs in the data block for the first time. For later updates, the same log block is utilized. In other words, there is one-to-one mapping relationship between a data block and a log block. A log block may also contain *invalid pages* if the same logical address is asserted more than twice. A data block or a log block is called *full* if it has no more *clean* pages. If a logical address of a write operation is mapped to a data block or a log block which is full, then an extra operation called *the merge operation* is performed to make room for the new data. The *merge operation* takes one of free blocks and copies valid pages from the data block and the corresponding log block. Then, BAST erases the data block and the corresponding log block and returns them to the pool of free blocks. The *merge operations* incur a large amount of copying and erase operations, which critically degrades the overall performance of NFSDs.
- FAST (Fully-Associative Sector Translation) [4] is similar to BAST. However, it resolves the block-thrashing problem and frequent *merge operations* of BAST by introducing fully associative random log blocks and sequential log blocks. An analogy can be found in the conventional cache design. BAST and FAST correspond to the direct-mapped cache and the fully associative cache, respectively. Hence, FAST incurs more computing overhead than BAST.
- Superblock FTL [5] can be considered as a trade-off between BAST and FAST. Both FTLs are two extreme cases in managing log blocks. On the other hand, Superblock FTL partitions into several groups called *superblocks* rather than a single large group as is done in FAST. A super block is associated with only part of data blocks rather than all data blocks. According to the aforementioned analogy, Superblock FTL corresponds to the N-way set-associative cache, which is a trade-off between the direct-mapped cache and the fully associative cache. It also provides the full associativity among the pages within a superblock.

C. Critical Design Parameters

Compared to FTL-SSs, the FTL-MMs have an additional issue – the mapping between a logical address and NFMs. The mapping scheme should well distribute the incoming write data to maximally exploit the channel and way concurrency. We can achieve the maximum concurrency by splitting the data into as many pieces as the number of NFMs (the product of the number of channels and the number of ways). The problem occurs when the length of incoming data is shorter than the page length. In that case, we waste a lot of storage space, since only a small portion of a valid page will be used for storing the data. The waste of storage space will eventually incurs more *merge operations* later, since the log blocks and data blocks will be used up quickly. For this reason, we do not consider the data splitting when the data length is shorter than the length of a page.

On the other hand, the log blocks in each NFM should be consumed in a similar rate. Otherwise, a particular NFM (or NFMs) will use up its log blocks quickly and incurs the *merge operations* frequently. In other words, the performance degradation due to the *merge operations* is maximally postponed when all NFMs have the identical log block consumption rate.

The other important parameter is scalability. Even though more channels and more ways will improve the overall bandwidth, this does mean that their utilization is also improved. For this reason, the general method for designing an FTL-MM should generate an FTL of which channel utilization ratio is scalable with respect to the number of channels and ways.

D. Address Translation Scheme

Fig. 3 (a) shows the conventional structure of a 32-bit logical address. It consists of three fields – block address, page address, and sector offset. The blocks and page addresses are used to find the corresponding physical data block and page from the address-mapping table, and the sector offset is used to represent the offset within a page. Along with the logical address, the host system sends the data length in terms of sectors and the operation mode (read or write).

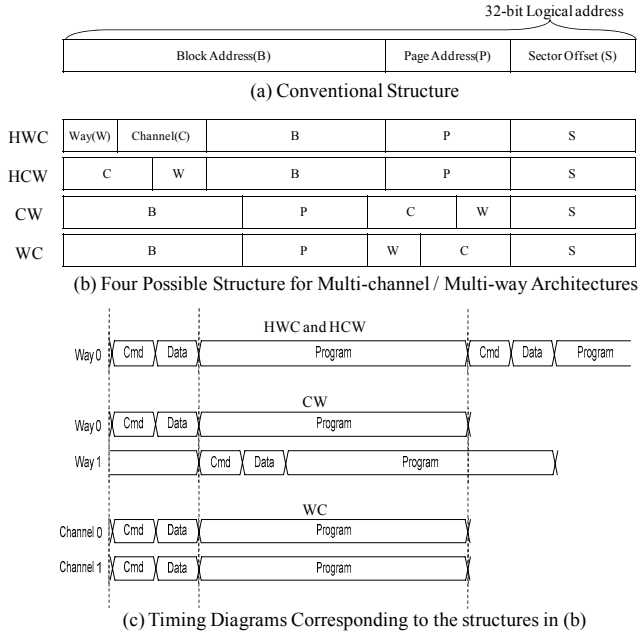


Fig. 3. Structures of 32-bit logical address and timing diagrams corresponding to the structures.

In the multi-channel / multi-way architecture, it is necessary to transform the conventional logical address structure to handle the channel and way mapping issue. For this reason, we add two extra fields called *C field* and *W field*. Four possible choices with these fields are depicted in Fig. 3 (b).

No matter what structure is chosen, the bit width of the B field should be reduced for allocating part of its bits to the two additional fields. An NFSD with *CH* channels and *WY* ways requires log_2CH bits and log_2WY bits the fields C and W, respectively. Even though the bit width of the B field is reduced, there is no problem to represent all the blocks, since a block is selected by the B field in conjunction with the channel field and the way field. Note that the channel and way fields do not change the capacity of the block address representation. They only alter the mapping relationship between the logical addresses and the data blocks.

Next, a qualitative performance comparison of these four structures is shown in Fig. 3 (c). In this example, two consecutive write operations are performed in the two-channel / two-way architecture. The data length of each write is one page, and their logical addresses are sequential. When both fields are allocated in high order bits (HWC and HCW structures), two consecutive write operations are mapped to the same NFM, hence there is no way to process them concurrently. On the other hand, the other two structures (CW and WC) allow two consecutive write operations in parallel. The WC structure is preferred, since the channel striping completely overlaps two write operations. This example clearly shows that the multi-channel / multi-way architecture will be benefited the WC structure.

The address translation scheme may not be advantageous for a random write pattern that is not the major focus in multi-channel / multi-way architectures. On the contrary, the performance enhancement for random write patterns is actively researched in different aspects such as transaction scheduling and DRAM cache buffering [10]. Note that the proposed method is complementary to these techniques, since it simply changes the address translation scheme without generating any side effects.

E. Address Mapping Table

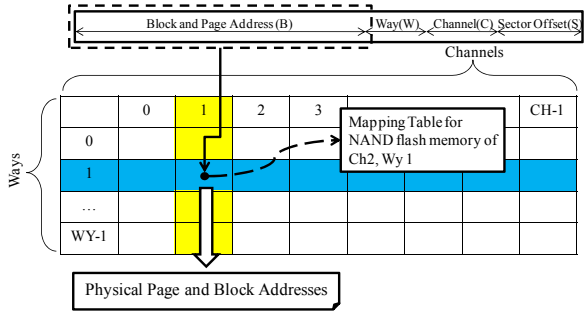


Fig. 4. Address-mapping table for multi-channel / multi-way NFSDs.

We emphasize the scalable design of a address-mapping table. More specifically, we design the mapping table to keep the nature of the FTL-SS as much as possible. One obvious way to achieve this is to minimize the inter-play among the NFMs. In other words, we do not allow the sharing of resources among NFMs except for channels. Without this policy, large overhead can occur and severely degrade the overall performance as shown in Example 2.

Example 2: each NFM has its own log blocks which can be accessed by other NFMs. If a page copying occurs across the ways, a page is read from one NFM by the NFM controller and then sent to the other NFM. The overall latency of this procedure is much longer than a page copying in a single NFM, since an NFM generally supports such a feature. Even worse, the transfer will include the data transfers among the NFM controllers and main memory if a page copying occurs across the channels. This scenario indicates that the resource

sharing may incur large performance overhead. Moreover, the performance is not scalable and not predictable with respect to the number of channels and the number of ways.

The non-sharing policy is implemented as a two-dimensional block array table as shown in Fig. 4, where each cell corresponds to a mapping table of each NFM. The structure of the mapping table of each cell is identical to that of the FTL-SS. The way field and the channel field are used as the row index and the column index of each cell, respectively. For the selected cell, remaining bits of the logical address determine the appropriate block and page.

As mentioned above, the scalability is one of the benefits of this structure. In addition, it has the following advantages. First, the computation overhead is marginal, since the additional computation, compared to the single-channel / single way FTL, is just the channel and way indexing. Second, the design time from the single-channel / single-way FTL is also marginal, since it just requires expanding a block mapping table into a two-dimensional array. Third, the structure is general enough to be applicable to any block-mapped FTLs, since it guarantees the atomicity of the single-channel / single-way address-mapping table by simply adding a layer on top of it. With these advantages, the proposed scheme can maximize the utilization of the given hardware resources (*i.e.,* channels and ways) without causing any side effects.

IV. Experimental Results

A. Experimental Setup

We implemented a trace-driven simulator for each FTL discussed in Section III. For each simulator, it is possible to configure some hardware parameters such as the number of channels, the number of ways, and NFM timing specification. We adopted the timing specification of a commercial NFM [2] which is shown in TABLE I.

TABLE I
Basic Latency Information of SLC NAND Flash Memory

Operation	Latency (μsec)
Page write operation	251.5
Page read operation	76.5
Block erase operation	1500.2

TABLE II
Information of Used Traces

#	Name	The total number of Requests	Average Write Length (Pages)	Details
1	distilled	300,000	4.02	FAT32 / Various PC usage
2	Download	3722	10.06	FAT32 / Large file downloads
3	Synthetic Ran	300,000	0.56	Synthetic Trace / Random
4	Synthetic Seq	50,000	16.00	Synthetic Trace / Sequential

We configured each simulator to target an NFSD whose total capacity is 16GB. It also has 65536 blocks and each block has 4KB-sized 64 pages. Based upon this setting, we measured the performance of FTLs, while varying the number of channels and ways. We consider five different architectures – 1x1, 2x1, 2x2, 4x2, and 4x4 architectures, where the first number and the second number indicate the number of channels and the number of ways, respectively. We also set the operating frequency of a channel to 64MHz at which the channel will be fully utilized by four interleaved writes according to the specification of TABLE I.

TABLE II summarizes the traces we used in this experiment. The first two traces are real traces collected from real systems. Trace 1, "distilled" was collected by running several programs on a PC [6]. Trace 2, "Downloads" was collected using DiskMon [7], while a large file is being downloaded from the Internet to a 64GB solid-state disk whose file system was FAT32.

One the other hand, Trace 3 and 4 were synthetic traces. The address sequences of both traces were created based on random number generation. In case of Trace 4, the data size per request was set to 128 sectors to measure the efficiency of FTLs for sequential data access patterns. The data size per request in Trace 3 was set smaller than or equal to the page size, and it varied from 1 sector to 8 sectors to appreciate their efficiency for random short data patterns. The fourth column in TABLE II shows the average data size per request for each trace. These numbers implies the utilization of channels and ways. More specifically, the trace with a larger number will be more benefited by the increase of channels and ways, since it is possible to split the data over the channels and ways.

B. Oracle FTL

For comparison purpose, we introduce an ideal FTL called *oracle FTL*. It is only realizable with the offline analysis of the target trace, since it assumes that the future of the trace is known and that there is no computing overhead. Hence, it cannot be implemented in practice. The *oracle FTL* allocates a pair of channel and way to each data such that the allocation never induces any garbage collection. For this reason, it can maximally utilize the given hardware resources. Note that the *oracle FTL* does not split the data when its size is smaller than a page as the proposed method. In other words, it sets the performance upper bound of FTLs.

C. Comparison metrics

To measure the impact of our method on performance, we define several metrics as follows:

- **Throughput** is a metric to measure the quantity of data processed in a second for a given architecture and trace pair. Its unit is MB/s. This metric is used for evaluating the performance of FTLs and address translation schemes addressed in Section III in absolute value.
- **Channel utilization (CU)**: is a metric to assess how efficiently channels are utilized. It is defined as the throughput of an FTL normalized to the sum of channel

bandwidths. Note that the sum of channel bandwidths means the maximum bandwidth provided by all physical channels. Hence, it is proportional to the number of channels.

- **Oracle-aware channel utilization (OCU)**: is defined as the CU of an FTL over the CU of the *oracle FTL* for comparing the performance of an FTL with that of the *oracle FTL*.
- **Relative oracle-aware channel utilization (ROCU)**: is a metric to evaluate the performance scalability of FTLs. In our work, it is desirable that the OCU of an FTL-MM is greater than or equal to that of FTL-SS, this means that the increased hardware resource (channels and ways) is utilized more efficiently than (or as effectively as) in FTL-SS. For this purpose, we define the ROCU of an FTL-MM as its OCU over the OCU of the corresponding FTL-SS.

D. Comparison of Address Translation Schemes

We extended the three FTL-SSs (BAST, FAST, and Superblock FTL) for the 4-channel / 4-way architecture by using our method. For each FTL-MM, we also created four variants and they employ the address translation schemes addressed in Section III.D. The performance comparison of these variants is shown in Fig. 5. We omit two schemes – HWC and HCW in this comparison, since they cannot exploit the concurrency for two sequential writes as shown in Fig. 3. The black and white bars correspond to WC and CW, respectively. WC always shows higher throughput for all FTL-MMs. The channel striping achieves higher throughput compared to the way interleaving by complete overlapping of writes. The advantage of using WC decreases when the data size per request is short as shown in Fig. 5 (c), since the overlapped writes rarely occur.

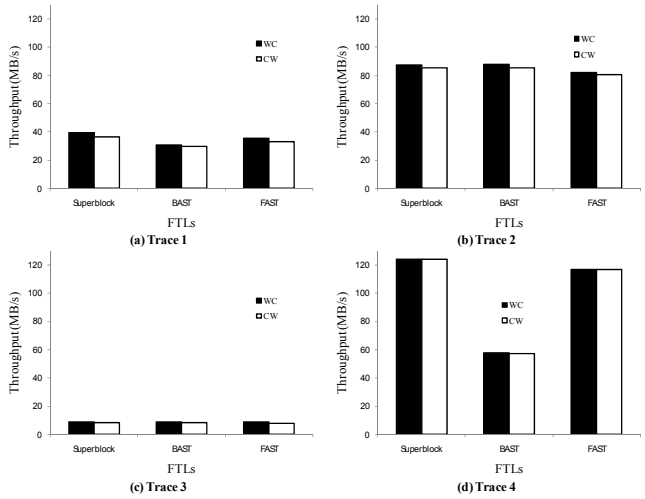


Fig. 5. Throughput of an 4-channel / 4-way architecture using three FTLs with two different bit positioning for four traces.

E. Channel Utilization of Oracle FTL

We show the CUs of the *oracle FTL* for four traces in Fig. 6. Even though the *oracle FTL* is ideal, it cannot fully exploit the channel bandwidth, since its CU depends on the data length.

First, we compare the CUs for all traces at the 1x1 architecture. The *oracle FTL* achieves the highest CU for Trace 4, while showing the lowest CU for Trace 3. More precisely, the ranks of traces in terms of CU are closely related to their average data size which is shown in Table II. The larger the data size is, the higher the CU is. The frequency of the write operations with longer data is lower than that of the write operations with shorter data to store the same amount of data. Hence, shorter data writes waste more time for the Cmd and Data phase.

Next, we examine the CUs at other architectures. If the number of ways increases, while fixing the number of channels, the CU will increase until the channel is fully saturated. The increasing ratio critically depends on the data length. In the opposite case, the CU will decrease if the data length is not long enough to saturate the channels. If the data length is long enough, the CU will be unchanged. If both numbers change at the same time, the CU will be placed between the two boundary values.

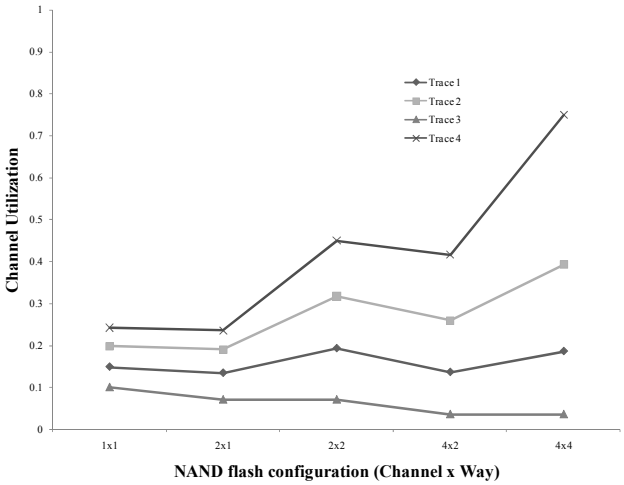


Fig. 6. The channel utilization of the *oracle FTL* for four traces.

F. Channel Utilization of Real FTLs

We compare the CUs of real FTLs with that of the *oracle FTL* in Fig. 7 through Fig. 9. The comparison was performed for all traces with various architectural choices, while varying the portion of log blocks over the entire blocks (log ratio) from 5% to 15%. The log ratio is not critical in a multi-channel / multi-way architecture except for BAST. 5% of log blocks are enough for reasonable performance as in for the single-channel / single-way architecture [5]. The different trend observed in BAST is mainly due to its low block-level associativity rather than the architectural effect, since it shows a large discrepancy depending on the log ratio at the 1x1 architecture for Trace 4. Other than the above case, the CUs of all FTLs well follow the CUs of the *oracle FTL*, even though their gap enlarges as the number of channels and/or ways increases. Therefore, the FTLs may have the performance scalability issue. For the quantitative analysis of the scalability issue, we compute ROCU for each FTL and summarize the results in TABLE III through TABLE V. We omit the results when the log ratio is 15%, since the results are similar, when the log ratio is 10%.

ROCU represents the OCU of an FTL at a specific architecture relative to that of the FTL at the 1x1 architecture. It clearly indicates the scalability of an FTL with respect to the number of channels and ways. The average ROCUs of FAST

and Superblock FTL for the all combinations of traces and log ratios are higher than 0.89. That is, the average performance degradation of both FTLs is only 10% in the worst case.

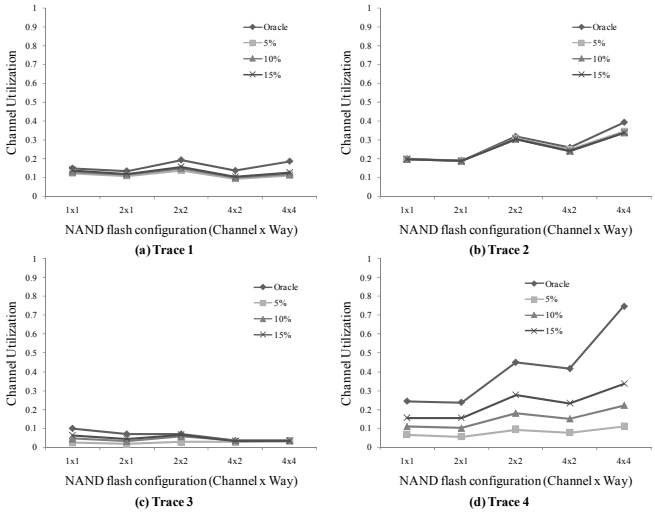


Fig. 7. Channel Utilization of BAST

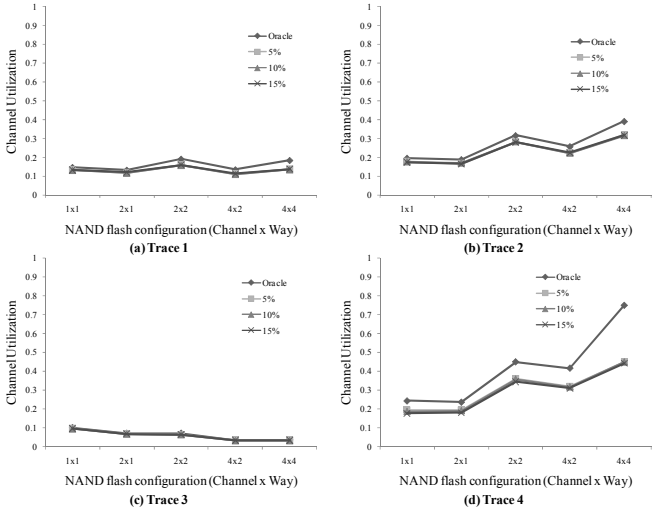


Fig. 8. Channel Utilization of FAST

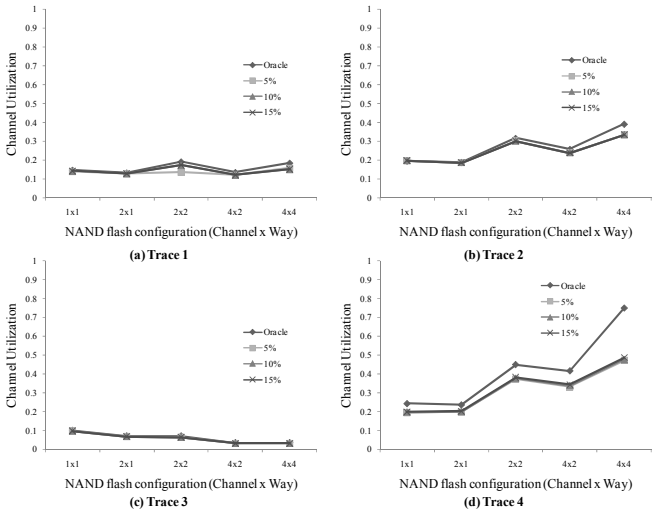


Fig. 9. Channel Utilization of Superblock FTL

Even in BAST, its average ROCU is not far behind those of FAST and Superblock FTL except for Trace 4. The log blocks are rapidly used up and a large amount of garbage collections eventually occur, due to the low block-level associativity of BAST and the sequential characteristics of Trace 4.

Some ROCUs at specific architectures and traces are larger than 1, meaning that the FTL-MM manages data and hardware resources more efficiently than its corresponding FTL-SS. This is due to the concurrent executions of erase and read/write operations on different NFMs. More precisely, an NFM can serve a write or read operation, while another NFM on a different channel (or way) is erased. We call this behavior *hidden erase*. The *hidden erase* is maximized for random short data patterns, which are exactly the case for Trace 3. It is also more effective to the FTLs which incur a lot of erase operations like BAST.

TABLE III
ROCU OF BAST

TRACE #	LOG RATIO(%)	ARCHITECTURES				
		2x1	2x2	4x2	4x4	AVERAGE
1	5	0.97	0.85	0.83	0.71	0.84
	10	0.98	0.88	0.84	0.73	0.86
2	5	1.00	0.98	0.96	0.88	0.96
	10	0.99	0.96	0.93	0.86	0.93
3	5	0.97	1.64	3.23	3.82	2.42
	10	1.00	1.82	2.18	2.18	1.80
4	5	0.85	0.76	0.67	0.54	0.71
	10	0.95	0.89	0.80	0.66	0.83

TABLE IV
ROCU OF FAST

TRACE #	LOG RATIO(%)	ARCHITECTURES				
		2x1	2x2	4x2	4x4	AVERAGE
1	5	0.99	0.92	0.91	0.82	0.91
	10	1.27	1.17	1.08	0.85	1.09
2	5	0.98	0.98	0.95	0.90	0.95
	10	1.02	1.00	0.97	0.90	0.97
3	5	1.00	0.95	0.97	0.97	0.97
	10	1.00	0.95	0.97	0.97	0.97
4	5	1.03	1.00	0.96	0.75	0.93
	10	1.05	1.03	0.99	0.78	0.96

TABLE V
ROCU OF SUPERBLOCK FTL

TRACE #	LOG RATIO(%)	ARCHITECTURES				
		2x1	2x2	4x2	4x4	AVERAGE
1	5	1.00	0.74	0.94	0.89	0.89
	10	1.00	0.94	0.93	0.85	0.93
2	5	0.99	0.96	0.93	0.86	0.93
	10	0.99	0.96	0.93	0.86	0.93
3	5	1.00	0.90	0.89	0.89	0.92
	10	1.00	0.90	0.89	0.89	0.92
4	5	1.04	1.03	0.99	0.78	0.96
	10	1.04	1.03	1.00	0.78	0.96

G. Distribution of Erase Operations

Uniform distribution of erases over NFM cells is crucial to prolong the lifetime of an NFM. Similarly, the distribution uniformity over the NFMs is important in multi-channel / multi-way architecture.

Fig. 10 shows the erase count distributions of Superblock FTL at the 4x4 architecture (log ratio = 10%). Fig. 10 (b) for Trace 2 is not our concern, since the total erase counts is too small. Fig. 10 (c) and Fig. 10 (d) show the uniform

distributions for Trace 3 and Trace 4, respectively. Even though their data sizes are quite different, their address sequences are random. These results indicate that the erase count uniformity is strongly dependent on the locality of the address sequences rather than the data length. The distribution shown in Fig. 10 (a) further supports this claim, since the distribution is less uniform than it is in Fig. 10 (c) and Fig. 10 (d) due to the locality of the address sequence of Trace 1. Even though we have shown the analysis of the erase count distribution over the NFMs, our method does not tackle this issue directly, which we will investigate as future work.

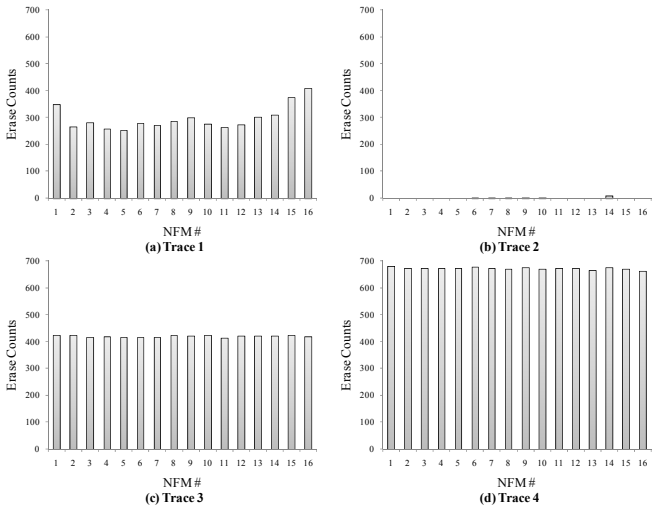


Fig. 10. Erase Counts of 16 NFMs at the 4x4 architecture by Superblock FTL.

V. CONCLUSION

We proposed a general method to design FTL-MMs. We proved its generality by applying it to three well-known FTL-SSs – BAST, FAST, and Superblock FTL. The advantage of our method is that it requires little design time to extend from FTL-SSs. Additionally, its implementation scheme incurs little computation overhead for indexing blocks. The extensive experimental results we obtained show that the FTL-MMs generated by our method are comparable to the *oracle FTL*. It has been also shown that the FTLs are scalable with respect the number of channels and ways from the performance perspective. In addition, we analyzed the erase count distribution over the NFMs for the first time. As future work, we will extend the proposed method to consider the address locality in the address translation scheme for improving the distribution uniformity of erase counts.

REFERENCES

[1] J.U. Kang, J.S. Kim, C. Park, H. Park, J. Lee, "A multi-channel architecture for high-performance NAND flash-based storage system," Journal of Systems Architecture, vol.53, Issue 9, pp. 644-658, Sep. 2007.

[2] Hynix Semiconductor Inc., HY27UG088G(5/D)B Series 8Gbit (1Gx8bit) NAND Flash Rev0.2, http://www.hynix.com/datasheet/pdf /flash/ HY27UG088G(5_D)B (Rev0.2).pdf, January 2008.

[3] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *Consumer Electronics, IEEE Transactions on*, vol.48, no.2, pp.366-375, May 2002.

[4] S.W. Lee, W.K. Choi, and D.J. Park, "FAST : An efficient flash translation layer for flash memory," EUC Workshops 2006, pp.879-887, 2006.

[5] J.U. Kang, H. Jo, J.S. Kim, J. Lee, "A superblock-based flash translation layer for NAND flash memory," EMSOFT'06, Oct. 2006.

[6] L.P. Chang and T.W. Kuo, "An adaptive striping architecture for flash memory storage systems of embedded systems," *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE* ,pp. 187-196, 2002

[7] Mark Russinovich, DiskMon for Windows v2.01, http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx, Nov. 2006.

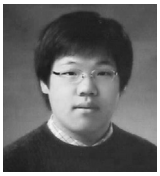
[8] C. Park, Talawar, P., D. Won, M.J. Jung, J.B. Im, S. Kim, and Y. Choi, "A High Performance Controller for NAND Flash-based Solid State Disk (NSSD)," *Non-Volatile Semiconductor Memory Workshop, 2006. IEEE NVSMW 2006. 21st* , vol., no., pp.17-20, 12-16 February 2006.

[9] Jim Cooke, "Flash Memory Techonology Direction," WinHEC 2007, May 2007.

[10] H. Kim, S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage,", 6th USENIX Conference on File and Storage Technologies, 2008.

[11] H. Kim, and S. Lee, "A new flash memory management for flash storage system," in Proc. Computer Software and Applications Conference, pp. 284-289, 1999.

[12] M.L Chiang, P. C.H. Lee, R.C. Chang, "Cleaning policies in mobile computers using flash memory," Journal of Systems and Software, vol.48, no.3, pp.213-231, 1999.



Sang-Hoon Park received the B.S. degree in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2009. He is currently a M.S. candidate in Yonsei University. His research interests include System on Chip, NAND flash based mass storage architecture and system architecture.

Seung-Hwan Ha received the B.S. degree in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2008. He is currently a M.S. candidate in Yonsei University. His research interests include System on Chip, NAND flash based mass storage architecture and system architecture.

Kwanhu Bang (S'06) received the B.S. degrees in computer science and in electronic engineering and the M.S. degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2006 and 2008, respectively. He is currently a Ph.D. candidate in the School of Electrical and Electronic Engineering at Yonsei University. His research interests include bio-computation, flash memory applications, and system-level low-power design.

Eui-Young Chung (S'99-M'06) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2002. From 1990 to 2005, he was a Principal Engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. He is currently an Associate Professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea. His research interests include system architecture, bio-computing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low power applications and flash memory applications.