Storage Architecture and Software Support for SLC/MLC Combined Flash Memory

Soojun Im and Dongkun Shin
Sungkyunkwan University
Suwon, Korea
{lang33, dongkun}@skku.edu

ABSTRACT

We propose a novel flash memory management software for SLC/MLC combined flash memories which are recently introduced to provide flexible and cost-efficient embedded storage systems. To provide a fast and large capacity of flash memory, the proposed scheme utilizes the SLC area as log buffer and the MLC area as data block. Considering the high write cost of MLC flash, the garbage collection for the SLC log buffer moves a page into the MLC data block only when the page is cold or the page invokes a small migration cost. We also propose the bypassing technique which sends a large sequential data into the MLC flash directly not through the SLC log buffer. From the experiments, we can know that the proposed scheme utilizes the SLC log buffer effectively providing better performance compared with the previous flash management schemes for the SLC/MLC combined flash. [1]

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**] *real-time and embedded systems*.

J.7 [**Computers in Other Systems**]: *consumer products.*

General Terms

Algorithms, Design.

Keywords

flash memory, flash translation layer, SLC/MLC combined flash, embedded system, storage system

1. INTRODUCTION

NAND flash memory has been used as a non-volatile storage device for mobile embedded systems (such as MP3 players, PDAs and digital cameras) because of its low-power consumption, high random access performance and high mobility. There has recently been a significant growth in the NAND flash market due to the increase of MP3 player and digital camera since these devices should store a large amount of multimedia data.

There are two types of flash memories: single-level-cell (SLC) and multi-level-cell (MLC). In SLC flash memory, one flash memory cell represents one bit. In MLC flash memory, more than one bit can be represented using multiple voltage thresholds [1]. SLC flash memory is faster, more reliable and has a larger number of erase cycles than MLC. However, MLC flash provides a larger storage capacity than SLC flash for the same-sized die, thus it is cheaper than SLC flash. So, MLC flash is a promising solution for large-scale flash memory systems such as USB flash memory or solid-state disk (SSD).

Recently, SLC/MLC combined flash architectures are introduced. One is to use both SLC flash chip and MLC flash chip to compose a large scale SSD [2]. In this architecture, there are multiple flash memory chips in an SSD and some portion of chips are SLC flash chips and the other portion of chips are MLC flash chips. The other approach is to use SLC/MLC combined flash chip which has both SLC blocks and MLC blocks in a single chip. For example, there are Samsung's Flex-OneNAND [3] and Toshiba's mobileLBA-NAND [4]. In these flash memory chips, flash memory blocks are divided into two areas: SLC area and MLC area. Depending on the size of each area, the total storage capacity of flash memory is changed. For example, if there are 1024 blocks in a flash memory chip where 256 blocks are SLC blocks and 768 blocks are MLC blocks, the total capacity is 458MB (=65MB+393MB) when we assume the block sizes in SLC and MLC are 256KB and 512KB, respectively.

Table 1. Characteristics of SLC and MLC Flash Blocks

	SLC	**MLC**
Page size	4KB	
Block size	256KB (64pages)	512KB (128pages)
Page read	45us	50us
Page write	240us	1ms
Block erase	500us	
P/E cycles	50K	10K

Table 1 shows the characteristics of SLC flash block and MLC flash block of SLC/MLC combined flash chip. The page sizes of two types are same but they have different block sizes. While there are no significant difference between the read performances of SLC and MLC, the write performances are significantly

[1] This work was supported by the Koreas Research Foundation Grant funded by the Korean Government (MOEHRD). (KRF-2007-331-D00358)

different. The possible number of program and erase (P/E cycle) of SLC block is five times of MLC block. This data is taken from the data sheet of Samsung's Flex-OneNAND [3]. Note that the characteristics of SLC flash chip and MLC flash chip are different from the SLC area and the MLC area of SLC/MLC combined flash chip. For example, while the read performance of MLC chip is generally slower than that of SLC chip, the read performances of SLC area and MLC area of the combined flash are similar.

Flash memory has several special features unlike the traditional magnetic hard disk. First one is its "erase-before-write" architecture. To write data in a block, the block should be erased before. Second feature is that the unit sizes of erase operation and write operation are different. While the flash memory is erased by the unit of block, the write operation is performed by the unit of page. A block is a bundle of several pages. Third feature is that the page writing (programming) within a block should be done sequentially. That means that the LSB (least significant bit) page of the block should be programmed before the MSB (most significant bit) pages of the block. Random page address programming is prohibited. The feature is called SOP (Sequentiality of Programming).

Due to these features, special software called flash translation layer (FTL) is required, which maps the logical page address from the file system to the physical page address in flash memory devices. The FTLs can be divided into three classes depending on the address mapping granularity, i.e., block-level mapping, page-level mapping and hybrid mapping. In the block-level mapping, only a logical block address is mapped to a physical block address, and the same page offset in a block is used. So, a logical page should be written by the *in-place* scheme, which means a page is written at the fixed location of a block determined by the page offset in a block. The block-level mapping needs a small-sized block-level mapping table. However, when data at a specific page is to be modified, the specified block should be erased and the non-updated pages as well as the updated page should be copied into the new block. This constraint results in a high page migration cost.

In the page-level mapping, a logical page can be mapped by the *out-of-place* scheme, which means a page can be written at any physical page in a block. If an update request is sent for a logical page which is already written in flash memory, the page-level mapping writes the new data to a different clean page and invalidates the old page since flash memory page cannot be overwritten. To do that, it should manage the page-level mapping table, thus the mapping table size is inevitably large.

The hybrid mapping is a compromised version of page mapping and block mapping. In this scheme, all the physical blocks are divided into log blocks and data blocks. The log blocks are called *log buffer*. So, the FTL using hybrid mapping scheme is called as a log buffer-based FTL. While the log blocks use the page-level mapping scheme, the data blocks are handled by the block-level mapping. When a write request is sent to FTL, the data is first written into the log buffer and the corresponding old data in data block is invalidated. When the log buffer is full and there is no empty space, one of log blocks is selected as a victim and all the valid pages in the log block migrate into data blocks and the log block is erased to make a room for on-going write requests. This step is called *block merge*. There are three kinds of block merges: full merge, partial merge and switch merge [4]. The partial merge and switch merge can be possible only when all the pages are written by the in-place scheme in a log block. When a log block has all the pages of the associated data block, the switch merge is possible. But, if a log block has free pages, the partial merge should be used. The hybrid mapping invokes a lower page migration cost compared to the block-level mapping, but requires a smaller-sized mapping table compared to the page-level mapping.

In this paper, we propose a novel FTL architecture for SLC/MLC combined flash. The FTL is based on the hybrid mapping. It exploits the SLC flash block as a log buffer of MLC flash block. The main two techniques of the proposed FTL are garbage collection and bypassing. They optimize the I/O performance of SLC/MLC combined flash considering the large write cost of MLC flash block.

The rest of the paper is organized as follows. In Section 2, the related works are introduced. Section 3 describes the details of proposed FTL scheme. Experimental results are presented in Section 4. Section 5 concludes with a summary and future works.

2. RELATED WORKS

There have been many researches on log buffer-based FTL. The log buffer-based FTLs are divided into two kinds depending on the block association policy, i.e., 1:1 log block mapping (BAST) [5] and N:1 log block mapping (FAST) [6]. The block association policy means how many data blocks are associated with a log block. When a log block has a page which is included to a data block, we say that the data block is associated with the log block. In the BAST (block associative sector translation) scheme, a log block is allocated for only one data block. In the FAST (fully associative sector translation) scheme, a log block can have a page of any data block. However, if the write request pattern is random, the 1:1 mapping scheme will show poor performance since frequent log block merges are inevitable. This is called the *log block thrashing* problem.

To solve the problem of 1:1 mapping scheme, N:1 scheme was introduced. In N:1 scheme, a log block can be used for any data block at a time, thus reducing the number of block merges. The pages are written into a log block in the order of requests regardless of the corresponding data block number. Using the N:1 mapping, we can prevent the log block thrashing problem. However, the problem of N:1 mapping is its high block associativity. If we merge the log block with its associated data blocks, we should copy all the pages of the associated data blocks to another free data blocks. FAST maintains two kinds of log blocks, random log block (RLB) and sequential log block (SLB). A SLB is associated to only one data block. All the pages in the SLB are written by the in-place scheme. So, the SLB can be merged by either switch merge or partial merge.

Recently, LAST [7] FTL scheme further separates the log buffer into three partitions: sequential log buffer, hot random log buffer, and cold random log buffer. By utilizing the temporal locality and the spatial locality, LAST scheme improved the performance of garbage collection.

There are also N:K log block mapping techniques such as Superblock [8] and SAST [9]. In the N:K log block mapping, K number of log blocks are used for N number of data blocks. The N:K mapping is a compromised version of BAST and FAST.

3. STORAGE ARCHITECTURE

The SLC/MLC combined flash chip is originally proposed to store applications at the SLC area and data at the MLC area. However, since the reliability of MLC can be improved with an error correcting controller and the read performance of MLC area is similar to the SLC area, we think that it is better to use the SLC area as write buffer for the frequently-updated data. The address space provided to the file system is that of MLC area. The SLC area is used for log buffer of MLC area by log buffer-based FTL. Using these architectures, the SLC/MLC combined flash can provides the similar performance of SLC only flash chip with the similar capacity of MLC only flash chip.

In this paper, the write requests are first sent to the SLC area (log buffer), which is managed by the page-level mapping, by the log buffer-based FTL. The MLC area (data block) is managed by the block-level mapping since it generally has a large number of pages. A sequential large data bypasses the SLC area since they are generally write-once data. When there is no free space in the SLC area, the garbage collector is invoked. The garbage collector makes free spaces by reclaiming the space occupied by invalid pages or by sending cold data to the MLC area.

Park et al. [10] also proposed an FTL for SLC/MLC combined flash chip. However, the FTL does not use the SLC block as a log buffer instead it tries to store hot data at the SLC block and cold data at the MLC block to optimize the I/O performance. In addition, it uses a page-level mapping scheme for both the SLC area and the MLC area. So, it requires a large-sized mapping table. However, our proposed technique requires a small mapping table since it is based on the hybrid mapping scheme.

3.1 Garbage Collection

In the current log buffer-based FTL, *all* the valid pages of a victim log block are moved into the data block by block merge operation. So, it can be called block-level merge. However, we use a page-level merge considering the slow write performance of MLC area. If a page is frequently-updated, it is better to keep the page at the log buffer. So, the garbage collector transfers only the cold pages into the MLC area, thus the hot pages remain in the SLC area.

The garbage collector also considers the page migration cost. If only a small portion of a data block is updated, the updated pages are located at the SLC area and the non-updated pages are at the MLC area. Then, moving the page into MLC area will invoke a large amount of page copies within the MLC area to maintain the block-level mapping. In such a case, it is better to leave it at the log buffer until more write requests on the corresponding block arrive. So, the garbage collector selects the page to be sent to MLC area considering the number of valid pages in the corresponding MLC data block to be merged with the page.

The garbage collection is composed of three steps: data block classification, migration to data block and migration within log buffer.

3.1.1 Data Block Classification

First, we classify all the pages in SLC log block into three types: hot, warm and cold. If the update count of a page after the last garbage collection is large than the specified threshold value P_{hot}, the page is classified into the hot page. If a page is not accessed during more than P_{cold} number of garbage collections, the page is classified into the cold page. The other pages are warm pages. For example, in Figure 1, we assume that the pages 10, 20, 26, 27 and 28 are hot pages and the pages 11, 12 and 13 are cold.

Second, we classify all the MLC data blocks which are associated with the log blocks. The data blocks MB0, MB1, MB2 and MB3 are the associated data blocks. We estimate the numbers of hot, warm and cold pages of the data blocks. The number sequences (hot, warm, cold) of data blocks MB0, MB1, MB2 and MB3 are (0,6,0), (1,0,3), (1,2,0), and (3,1,0), respectively. If a data block has more than B_{cold} number of pages whose corresponding page in the log buffer is cold, the data block is regarded as cold block. If a data block has more than B_{hot} number of pages whose corresponding page in log buffer is hot and the data block is not cold block, the data block is regarded as hot block. Otherwise, it is considered as warm block. Therefore, if we assume that B_{cold} and B_{hot} is 2, then the data block MB1 is cold, the data block MB3 is hot, and the data blocks MB0 and MB2 are warm. The aim of this classification is to perform the page-level merge with only the cold block and the warm block.

3.1.2 Migration to Data Block

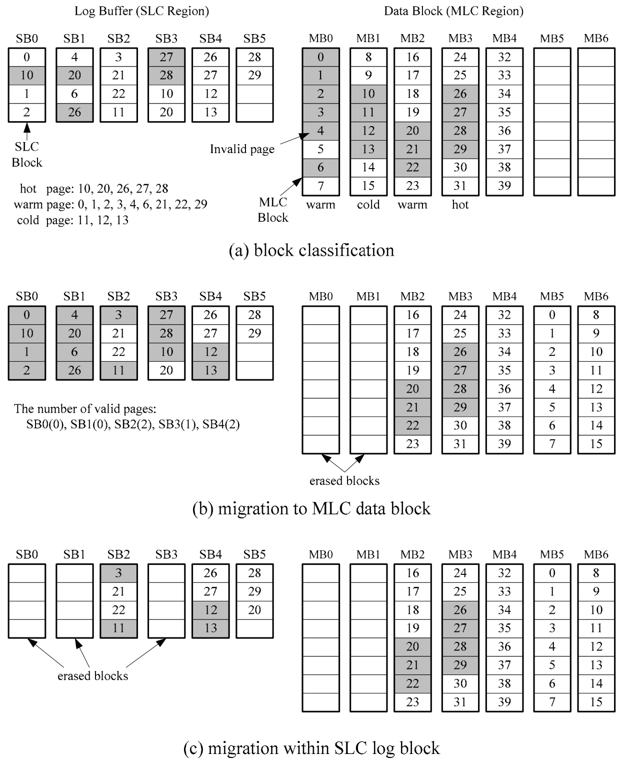
We move all the log pages whose associated data blocks are cold into data blocks. Since the cold block has a large number of cold pages, we evict the cold pages from the log buffer to make free spaces. For the warm block, we estimate the number of valid pages of the warm block. For instance, the warm data block MB0 has two valid pages while the warm data block MB2 has five valid pages in Figure 1(a). For the pages in log buffer whose corresponding data block is warm block with many valid pages, it is better to delay moving the pages into the MLC data block since the page migration will invoke a large number of page copies within the MLC blocks. So, if the number of valid pages in a warm block is more than θ, we does not merge the data block with log blocks. If we assume that θ is 4, only the data block MB0 is selected for block merge target. So, the logical pages 0, 1, 2, 3, 4, and 6 migrate into data blocks and they are invalidated at the log blocks. After the page migrations, the data blocks MB0 and MB1 can be erased as shown in Figure 1(b). Since the read performances of SLC and MLC are similar, we invalidate all the pages which migrate to the MLC block to reclaim the free space. If the read performances are quite different, it could be better not to invalidate the hot pages for the future read requests[2]. The value of θ determines the number of pages to migrate into MLC area. It also affects the frequency of garbage collection.

3.1.3 Migration within Log Buffer

After the migration to data block, there will be many invalid pages in the log buffer. Therefore, we should reclaim the free space by erasing some log blocks. If all the pages in a log block are invalid, we can erase the block and make it clean to be reused. However, if there are both valid pages and invalid pages in a log block, we should determine whether to move the valid pages into another log blocks and erase the block or not to reclaim the invalid pages of the log block. We should consider how many valid pages are there since it determines the page copy overhead. So, we estimate the number of valid pages in each log block. Only when the log block has a smaller than δ number of valid pages,

[2] Even though a page is hot, the page can be merged with a data block if the data block is cold or warm block.

we move all the valid pages of the log block into another log block. If we assume the value of δ is 2, only the page 20 is copied into the SB5 and the log blocks SB0, SB1 and SB3 are erased in Figure 1(c).



Log Buffer (SLC Region)
Data Block (MLC Region)
SLC Block
Invalid page
MLC Block
hot page: 10, 20, 26, 27, 28
warm page: 0, 1, 2, 3, 4, 6, 21, 22, 29
cold page: 11, 12, 13
warm
cold
warm
hot
(a) block classification
The number of valid pages:
SB0(0), SB1(0), SB2(2), SB3(1), SB4(2)
erased blocks
(b) migration to MLC data block
erased blocks
(c) migration within SLC log block

Figure 1 Garbage collection on SLC log buffer

3.2 Bypassing the log buffer

FAST scheme uses a sequential log block for the sequential write request. In the sequential log block, data is written by the in-place scheme. So, we can perform the switch merge which changes the sequential log block to a data block without page copy and block erase. However, in the SLC/MLC combined flash, if we allocate a sequential log buffer at SLC area, we cannot use the switch merge since log block and data block are located at different areas. Therefore, it is useless to allocate a sequential log block in SLC area. Instead, we use the bypassing technique for the sequential write request.

Generally, large and sequential data has a low temporal locality [7]. So, we make the large data to bypass the SLC log buffer. If the sector length of a write request from the file system is larger than the specified threshold α, the FTL sends it directly to the MLC area. In most of cases, the file system sends the write request of a large data to the storage after dividing it into multiple write requests. So, there will be subsequent write requests for a large-sized file. Since we cannot overwrite a flash memory page, even a small-sized update request invokes many page copies in the block-level mapping. So, to reduce the page copy cost, we use the *update block* as shown in Figure 2.

For example, if the write request write(1,4) (write four pages from the logical page offset 1) is sent to the FTL, we allocates an update block MB4 and writes the new five pages 0, 1, 2, 3 and 4 into the block and invalidate the corresponding old pages in MB0. Even though the page 0 is not updated, we copy it from MB0 to MB4 to preserve the in-place write scheme in MB4. Note that we cannot write the page 0 after the pages 1, 2 and 3 are written in MB4 due to the SOP feature of flash memory. The block mapping table maintains the physical block number (PBN) and the update physical block number (UPBN) for each logical block number (LBN). After the update for a data block, a corresponding update block is allocated and the UPBN is recorded into the block mapping table.

If the update block is fully written, it is changed to a data block and the old data block is erased to be reused. If the update request for the valid page in the update block arrives, the update block should be merged with the corresponding data block because only one update block can be allowed for a data block. We also limit the maximum number of update blocks. So, if there is no more update block to be allocated, one of update blocks should be merged with its corresponding data block. The victim should be selected considering the number of free blocks in the update block. Since we should copy pages as the number of free pages in the update block, we select the update block which has the smallest number of free pages.

To determine whether a write request bypasses the SLC log buffer or not, we use the request data length (L) and the page offset from the last update page (O). For example, the page offset of the first write request write(1,4) is 1. However, if another write request write(6,2) comes subsequently, the page offset from the last update page is 1 (not 6) since the last written page is 4. The larger the value of L is and the smaller the value of O is, the priority of bypassing should be larger. So, we can represent the bypassing condition as follows:

$$L \geq \alpha \text{ and } O \leq \beta$$

We can adjust the values of α and β to limit the number of write requests on SLC. If the SLC blocks are worn out too quickly, we decrease α and increase β to make more write requests to bypass the SLC log buffer. If the MLC blocks are worn out too quickly or the write requests for the MLC block are too many, α is increased and β is decreased. This technique is called *bypassing-throttling*. Using the bypassing-throttling, we can balance the number of program/erase cycles of SLC blocks and MLC blocks.

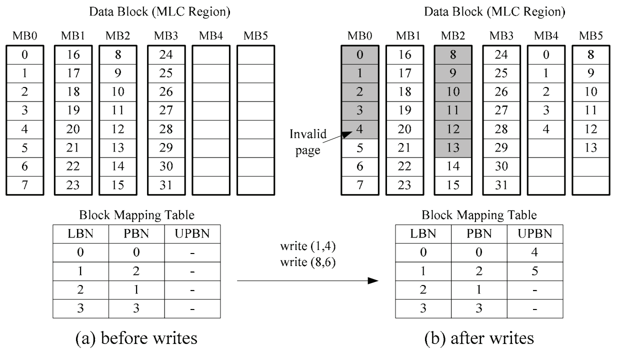


Figure 2 Using update block in the MLC block

4. EXPERIMENTS

We evaluated the performance of the proposed scheme using simulation. The workloads used for our experiments were extracted from Microsoft Windows XP-based desktop PC, running several applications, such as documents editors, music players, web browsers and games. We collected the trace from both FAT32 and NTFS file systems. We also collected other traces running IOzone and Postmark benchmarks. For the characteristic of SLC/MLC combined flash memory, we used the values in Table 1.

We first evaluated the effects of variables related to the hot/cold separation, P_{hot}, P_{cold}, B_{hot} and B_{cold}. For all experiments, the best values of P_{hot} are 0. That is, if there are more than or equal to one number of update on a page, we should consider the page is hot. Figures 3 and 4 show the effects of P_{cold}, B_{hot}, B_{cold} and δ. As we decrease the values of P_{cold} and B_{cold}, there are many cold pages and cold blocks, thus the number of page migration to MLC area increases. As we decrease the value of B_{hot}, the number of hot page migration to MLC area decreases. For FAT32 trace, the best values are 25, 0, and 12 for P_{cold}, B_{hot} and B_{cold}, respectively. The best value of δ is 40. If the value of δ is too small, there will be frequent invocations of garbage collection since many invalid pages are not reclaimed. If the value of δ is too large, there will be many page migrations within the SLC area.

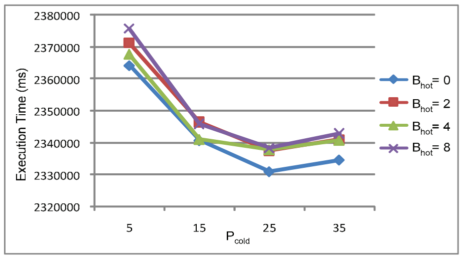


Figure 3 The effect of B_{hot} and P_{cold} (θ=64, δ=40, P_{hot}=0, B_{cold}=12, FAT32)

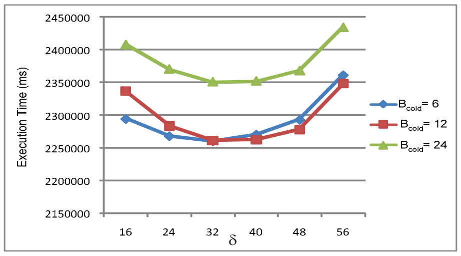


Figure 4 The effect of δ and B_{cold} (θ=64, P_{cold}=25, B_{hot}=0, FAT32)

We also evaluated the effects of θ. Figure 5 shows the execution time consumed by I/O. If θ is 0, none of the pages associated with the warm blocks migrates to MLC blocks. So, there are frequent garbage collector invocations as shown in Figure 6. As we increase the value of θ, the performance is improved because the number of garbage collector invocations decreases. However, if θ is larger than 64, there are performance degradations since the numbers of page migrations from MLC to MLC and from SLC to MLC per garbage collection increase as shown in Figure 6.

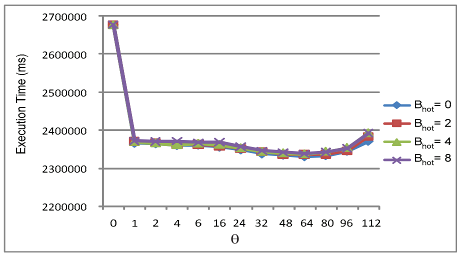


Figure 5 The effect of θ (δ=40, P_{cold}=25, B_{cold}=12, FAT32)

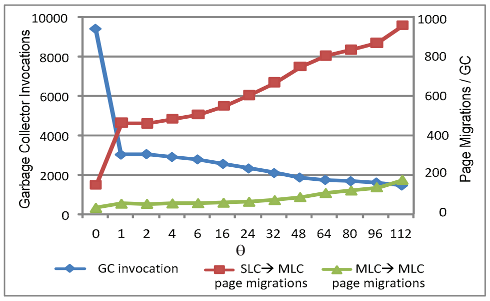


Figure 6 The garbage collections varying θ (δ=40, P_{cold}=25, B_{cold}=12, FAT32)

Figures 7 and 8 show the result of the trace collected on NTFS. The result patterns are similar to the trace of FAT32. However, the optimal value of θ is smaller than that of FAT32. This means that it is better to move the pages associated with the warm blocks since there are little temporal locality on the trace. For the same reason, the optimal values of P_{cold} and B_{cold} (15 and 6) are smaller than those of FAT32 (25 and 12). From the experimental results, we can know that the parameter values of P_{cold}, B_{hot}, B_{cold}, θ and δ determine the numbers of garbage collection invocations, page migrations within SLC area and page migrations from SLC to MLC. These values should be selected considering the workload patterns.

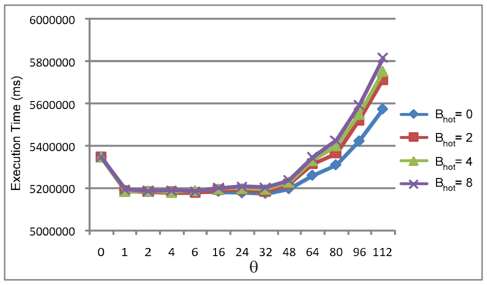


Figure 7 The effect of θ (δ=40, P_{cold}=15, B_{cold}=6, NTFS)

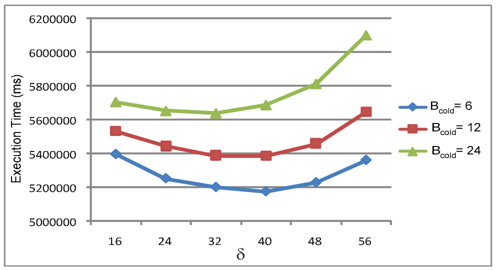


Figure 8 The effect of δ. (θ=32, P_{cold}=15, B_{hot}=0, NTFS)

Figure 9 shows the changes of normalized execution time varying the size of SLC area. As the number of SLC blocks (N_{SLC}) increases, the execution time decreases since the garbage collector is invoked infrequently. However, there is a significant difference between FAT32 trace and Postmark benchmark. The FAT32 trace has a high temporal locality. Our proposed FTL utilizes the locality well and provides a good performance even when the SLC size is small. So, there is little performance change as the size of SLC area increases. However, there are little temporal locality on the Postmark benchmark. So, the garbage collection is invoked frequently if the SLC size is small. The number of accesses to MLC area is significantly reduced as the SLC size increases.

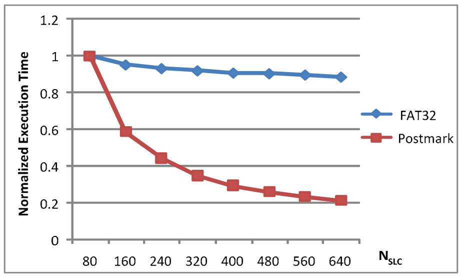


Figure 9 Performance comparison varying the size of SLC area (θ = 64, δ = 40, P_{cold} = 25, B_{cold} = 12, B_{hot} = 0)

We compared the proposed FTL scheme with the previous FTL schemes BAST and FAST. All the FTL schemes use the SLC area as a log buffer. Figure 10 shows the normalized execution times for four traces, FAT32, NTFS, IOzone and Postmark. We experimented for two different SLC area sizes, N_{SLC}=80 and N_{SLC}=160. The proposed scheme shows better performances for all traces.

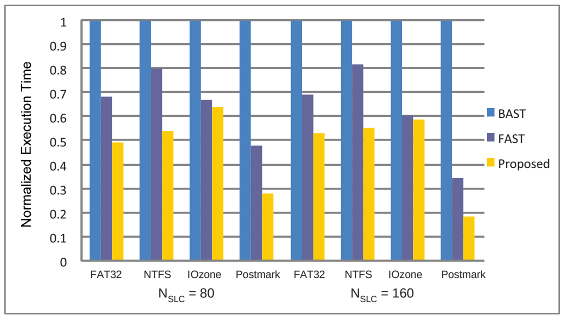


Figure 10 Comparison with previous FTL schemes

Lastly, we evaluated the effect of bypassing technique varying the parameters α and β explained in Section 3.2. Figure 11 shows the I/O execution time when the bypassing technique is used. Compared to the no-bypassing scheme, the performance is improved up to 7% using the bypassing technique. As we use a smaller value for β, the performance is more improved since the number of page migrations within MLC area is reduced. However, it is too small (for example, β=1), the benefit by the bypassing technique is reduced. Figure 12 shows that how many flash operations are reduced by the bypassing technique. When α=192KB, there is little changes on the SLC area, but the number of accesses on the MLC area is reduced. When α=64KB, the number of accesses on the SLC area is significantly reduced. However, the MLC area cost is increased due to the update block management overhead.

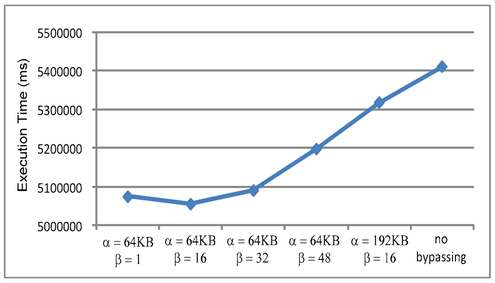


Figure 11 The effect of bypassing parameters

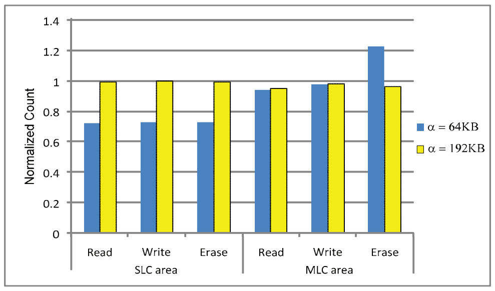


Figure 12 The number of flash read/write/erase counts normalized by those in no-bypassing technique

5. CONCLUSION

We have presented a novel flash memory management scheme for SLC/MLC combined flash chip. The proposed scheme is based on the hybrid mapping FTL and utilizes the SLC area as a log buffer. The garbage collection algorithm in the proposed scheme moves the pages in the SLC area into the MLC area selectively considering their localities and the migration costs. The bypassing algorithm sends a large sequential data into the MLC area to use the log buffer effectively. In the experiments, we showed the several parameters in the algorithm should be carefully determined considering the workload pattern. As a future work, we plan to study an adaptation scheme which adjusts the parameters used in the proposed scheme observing the workload pattern.

6. REFERENCES

[1] T. Cho et al. "A dual-mode NAND flash memory: 1-Gb multilevel and high-performance 512-Mb single-level modes," IEEE Journal of Solid-State Circuits, Vol. 36, Issue 11, 2001.

[2] L. Chang. "Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs," Proc. of Asia and South Pacific Design Automation Conference (ASPDAC), pp. 428-433, 2008.

[3] Samsung Electronics, 4Gb Flex-OneNAND M-die, http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_FlexOneNAND.html.

[4] Toshiba America Electronic Components, Inc., mobileLBA-NAND, http://www.toshiba.com/taec.

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

[6] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and H. J. Song. "A log buffer based flash translation layer using fully associative sector translation," ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, 2007.

[7] S. Lee, D. Shin, and J. Kim. "LAST: locality-aware sector translation for NAND flash memory-based storage systems," Proc. of SPEED'08, Salt Lake City, Utah, Feb. 2008.

[8] J. U. Kang, H. Jo, J. S. Kim, and J. Lee. "A superblock-based flash translation layer for NAND flash memory," in Proc. International Conference on Embedded Software, pp. 161-170, 2006.

[9] S. Y. Park, W. Cheon, Y. Lee, M.-S. Jung, W. Cho and H. Yoon. "A Re-configurable FTL (Flash Translation Layer) Architecture for NAND Flash based Applications," in Proc. of International Workshop on Rapid System Prototyping, pp. 202-208, 2007.

[10] S. H. Park, J. W. Park, J. M. Jeong, J. H. Kim and S. D. Kim. "A mixed flash translation layer structure for SLC-MLC combined flash memory system," Proc. of SPEED'08, Salt Lake City, Utah, Feb. 2008.