Increasing Multi-controller Parallelism for Hybrid-Mapped Flash Translation Layers

Hung-Yi Sung and Chin-Hsien Wu

Department of Electronic and Computer Engineering,
National Taiwan University of Science and Technology, Taiwan
{M10002137,chwu}@mail.ntust.edu.tw

Abstract. Nowadays, the architecture of solid-state drives (SSDs) is using multiple controllers to efficiently handle NAND flash memory chips. Several flash translation layers (FTLs) have been proposed to improve the overall performance of NAND flash memory. Therefore, the collaboration of FTLs and the multi-controller design of SSDs will become an important research topic. In this paper, we will propose a method to increase multi-controller parallelism for hybrid-mapped flash translation layers.

1 Problem Overview

In the paper, we explain the importance of handling read/write requests under a multi-controller design of SSDs. An SSD consists of a host interface, RAM buffer, a master controller, multiple slave controllers, and multiple NAND flash memory chips [1]. The master controller is responsible for the execution of a flash translation layer (FTL) which can handle read/write requests from the host interface. Each flash memory chip under a hybrid-mapped flash translation layer can be divided into data and log blocks. When a file system receives a read request, FTL will handle the read request and assign slave controllers to read data from the 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.

Compared to hard-disk drives (HDDs), SSDs could provide faster read and write operation time in terms of sequential and random data access. Currently, SSDs use a fixed architecture, called the multi-controller design [2]. Under the architecture, one controller can access to flash memory chips on its own bus which could reduce execution parallelism of multiple controllers. Therefore, the performance will decrease while many I/O requests access those chips which belong to the same controller, and other idle controllers cannot perform the I/O requests in different buses.

2 Design Concept

As shown in Fig. 1, when a request is coming, it contains the information of *Start_LPA* and *Size* which means "read/write the pages of length *Size* from the logical

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 567–570, 2014.

page address of *Start_LPA*". If it is a write request, the request will be divided into sub-requests by the block striping technique. Assume that the maximum size of one sub-request is one block. Sub-requests may include data blocks {D1, D2, ..., Di}, or log blocks {L1, L2, ..., Lj}. If it is a read request, FTL will search its mapping table and translate the read request to sub-requests that could include data blocks {D1, D2, ..., Di}, or log blocks {L1, L2, ..., Lj}. Then, the read sub-requests will be performed by reading the data or log blocks from the corresponding flash memory chips.

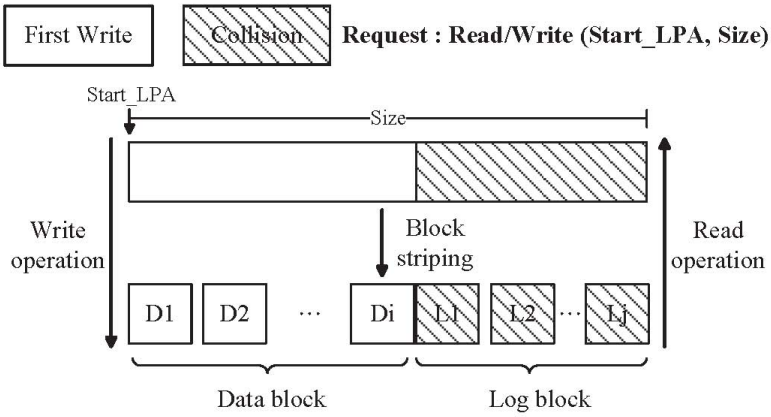

Fig. 1. Handling read/write operations

We show how to handle the first write request. As shown in Fig. 2.(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 the block striping technique. The best case is to write the data blocks {D1, D2, D3, D4} into different buses by the page striping technique. The worst case of Fig. 2.(a) happens when that only one controller is involved and sub-requests are written into one bus. 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 multi-controller parallelism.

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 log blocks {L1, L2, ..., Lj}. As shown in Fig. 2.(b), Request 2 will write data to two log blocks {L1, L2}. Assume that the corresponding data blocks of L1 and L2 are D1 and D2, respectively. If partial valid data are stored in both {L1, D1} and {L2, D2}, the best case is to write the log block {L1} to Bus 2 and the log block {L2} to Bus 3, where the corresponding data blocks {D1, D2} are also not locating in Bus2 and Bus 3. If the log block is located in the same bus with the corresponding data block, when valid data in {L1, D1} or {L2, D2} are required, only one controller can be involved and reduce the multi-controller parallelism, as shown in Fig. 2.(b).

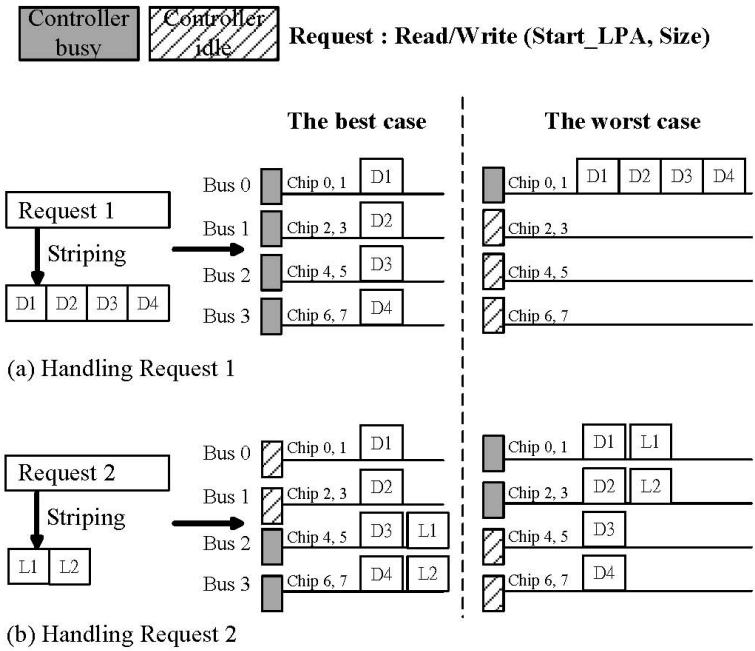

Fig. 2. Handling Request 1 and Request 2

Therefore, the allocation method of data and log blocks is to avoid a log block with its corresponding data blocks in the same bus. The allocation method can be implemented in the address translation function of FTL because FTL usually has a RAM-resident translation table, where each entry of the table contains the corresponding physical block addresses (i.e., data and log blocks) by indexing the logical block address. Assume that there are N buses and a write request can be translated to some sub-requests that could include data blocks {D1, D2, ..., Di}, or log blocks {L1, L2, ..., Lj}. For each data block Dx in {D1, D2, ..., Di}, it can be dispatched to the (x%N)-th bus in a block-striping way. For each log block Ly in {L1, L2, ..., Lj}, it can be dispatched to the bus where its corresponding data blocks are not locating. Therefore, if the number of corresponding data blocks is larger than the number of buses, a log block with the corresponding data blocks could co-exist in the same bus. In this case, the log block can be dispatched to the bus where its corresponding data blocks have the least number of valid pages. The design idea behind the allocation method is to distribute valid pages to different buses as much as possible. Thus, when read and write requests occur, we can increase the multi-controller parallelism to access flash memory chips efficiently.

3 Experimental Results

Four real traces are used in the experiments, as shown in Table 1. The improvement ratio with the proposed method for merge operations under 32 and 256 RW log blocks is shown in Table 2. According to the experimental results, we can utilize the execution parallelism to improve the execution time of partial and full merge operations.

Table 1. Four Real Traces

Trace	Total request Count	Total page accesses	Read ratio	Write ratio	Avg. read/write 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
AS SSD	246,957	8,925,678	33.37 %	66.63 %	69.96/ 29.10
Windows PC	2,398,728	8,532,159	55.24 %	44.76 %	3.54 / 3.58

Table 2. The improvement ratio with the proposed method for merge operations under 32 and 256 RW log blocks

Trace	32 RW log blocks	256 RW log blocks
Financial 1	15.52%	15.64%
Financial 2	15.68%	16.08%
AS SSD	3.9%	3.91%
Windows PC	8.34%	8.31%

References

1. Bez, R., Camerlenghi, E., Modelli, A., Visconti, A.: Introduction to Flash Memory. Proceedings of the IEEE 91(4) (April 2003)
2. Kang, J.U., Kim, J.S., Park, C., Park, H., Lee, J.: A multi-channel architecture for high-performance NAND flash-based storage system. Journal of Systems Architecture 53(9), 644–658 (2007)
3. Park, S.K., Park, Y., Shim, G., Park, K.H.: CAVE: channel-aware buffer management scheme for solid state disk. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 346–353 (May 2011)