International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 19 (2017) pp. 8937-8940

Performance Evaluation of Multi-threaded Flash Translation Layer

J. Kim[1] and I. Shin[1, a]

Graduate Student[1], Associate Professor[2]
[1]Department of Electronic Engineering, Seoul National University of Science & Technology, Nowon-gu, Seoul, South Korea.

[a]Orcid ID: 0000-0003-2819-0398

Abstract

Existing flash translation layer (FTL) operates mostly in a single-threaded manner. However, in recent years, the adoption of multi-cores inside SSD has been spreading, and it is common to connect several NAND flash chips in parallel structure. Therefore, if FTL operates with multi-threads, the SSD internal parallelism will be better utilized and the performance will be improved. In this study, we port uC/OS-II real-time operating system to a board equipped with NAND flash memory and implement a multi-threaded FTL. Performance evaluation results show that when the size of the I/O request queue is shallow, the average performance is almost the same as that of the single-thread FTL. However, as I/O queue depth increases, the average performance is improved up to about 5.5%.

Keywords: FTL, multi-threads, NAND flash memory, SSD

INTRODUCTION

Solid-state drives (SSDs) tend to employ multi-core internally connect multiple NAND flash chips in parallel to achieve the high performance. However, exiting flash translation layer (FTL) is implemented in single-threaded form, and thus when a single thread becomes blocked due to a specific event, other requests cannot be processed also. If FTL is implemented with multi-threads, the unnecessary delay can be avoided.

We have proposed a method to implement a multi-threaded FTL in our previous work [1]. The shared resources of FTL are the metadata of FTL, the mapping table, and NAND flash memory. We serialized the accesses to the shared resources with semaphores. However, because the multi-threaded FTL was implemented using win32 API on windows operating system, performance evaluation could not be performed on a board equipped with NAND flash memory. Only the implemented FTL operates correctly is confirmed [1].

This work aims at implementing the multi-threaded FTL in a board and evaluate its performance. For the purpose, we first build a board equipped with NAND flash memory and the ARM cortex processor. Then, uC/OS-II operating system is ported to this board. Finally, we create multiple tasks, and each task executes a FTL code. The shared resources are protected using semaphores supported by uC/OS-II.

A trace-driven performance evaluation shows that when the size of the I/O request queue is shallow, the average performance of the multi-threaded FTL is almost the same as that of the single-thread FTL. However, as I/O queue depth increases, the average performance is improved up to about 5.5%. The effect on the worst-case performance is not significant.

BACKGROUND AND RELATED WORKS

NAND flash memory is a type of EEPROM composed of multiple blocks. A block consists of multiple pages, and the page is the basic unit of read/write operation. A block is the unit of erase operation. A major constraint of NAND flash memory is that it does not support an overwrite operation. In other words, the target page must be clean in order to write data. The clean state means that data has not been written since the erase operation was performed on the page. Therefore, devices that use NAND flash memory as storage media support the overwrite operation through FTL. On a write request, FTL finds a new clean page and write the data to it. Thus, the current location of the data changes at each write, and therefore FTL maintains mapping information between the logical address space and the physical address space in memory.

FTL is classified into page mapping [2], block mapping [3], and hybrid mapping [4] according to the mapping unit. Among them, the page mapping FTL has a large memory requirement due to a large mapping table size, but shows the best performance. Therefore, most commercial SSDs are equipped with the page mapping FTL [5-7]. In this study, we also implement the page mapping FTL in a multi-threaded way.

[1] Corresponding Author

Meanwhile, SSDs connect multiple NAND chips in parallel using multiple channels. A chip is composed of multiple dies, each of which can perform NAND operations independently. Therefore, intensive studies have been conducted to improve SSD performance by utilizing the internal parallelism of SSD [5-7]. However, all of these studies were evaluated on the simulator, and the page mapping FTL was implemented in a single-threaded manner. This study differs greatly from that the multi-threaded FTL is implemented on real board and evaluated.

NAND Flash Memory Board

An open SSD board that can implement FTL is the Jasmine board [8]. However, since this board is designed to access NAND flash memory through a Waiting Room (WR) and there is only one WR, it is almost impossible to implement a multi-threaded FTL. Therefore, we create our own NAND flash memory board with ARM cortex processor (fig. 1). The board is equipped with 2GB of NAND flash memory, and the communication with the PC is performed through the UART. The SRAM in which the mapping table and metadata are stored is 8 MB, the number of pages per block is 128 KB, and the page size is 4 KB. Table 1 shows the major hardware specifications of NAND flash memory and the board.

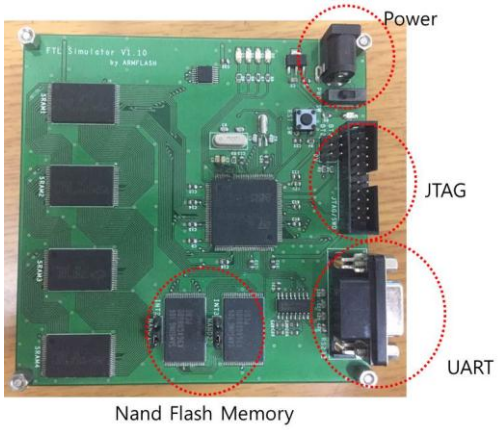

Figure 1: The board equipped with NAND flash memory

Table 1: Board properties

MCU	STM32F103ZTE6
SRAM	8MB
NAND capacity	4GB
Page size	4KB
Block size	512KB
Page read latency	60 us
Page write latency	800 us
Block erase latency	1.5 ms

Implementation of Multi-threaded FTL

To implement the multi-threaded FTL, we first ported the uC/OS-II operating system to the board. The uC/OS-II can create up to 64 tasks, and each task has different priority values. The communication between a host and the board is performed by the UART, and a queue is used to receive multiple I/O requests at a time.

In order to process I/O requests, we create five tasks as shown in fig. 2. Four tasks execute the FTL code, and the other task plays a role as a monitor process. It checks the status of NAND flash memory and wakes up the other tasks. In other words, task 0, task 1, task 2, and task 3 fetches an I/O instruction from the queue and process it one by one. If the task executes a long NAND flash memory operation such as page write or block erase, it calls the sleep() function to release the CPU to other task after sending the command to NAND. If all the tasks are in the sleep mode, task4 is executed. It checks whether the NAND operation is completed and wakes up the according task.

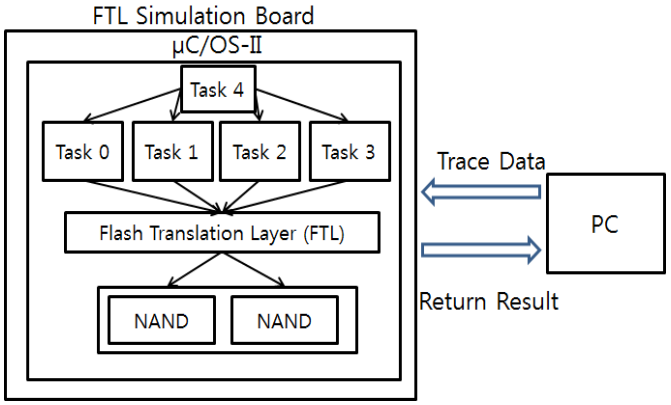

Figure 2: Multi-threaded FTL structure

Meanwhile, semaphores are used to protect shared resources. The main shared resources that must be protected are the metadata of FTL, mapping table, and NAND flash memory. When accessing the shared resources, the OSSemPend() function is called to acquire the semaphore. When exiting the critical section, OSSemPost() function is called to release the semaphore.

Meanwhile, each task uses its own working block. That is, working blocks are not shared between tasks. If tasks share a working block, even though each task writes data sequentially to the block, the sequential programming restriction can be violated by the task scheduling. Therefore, each task uses its own working block. Also, to minimize the waiting time for each task to wait for the NAND operation to complete, the task releases the CPU right after issuing the write or the erase operations as stated above. However, in the case of a read operation, which is completed relatively in a short time, the task waits while having the CPU until the read is completed.

In the case of the copyback operation, the task waits also until the operation is completed in order to prevent other task from interrupting in the middle of the copyback operation.

Performance Evaluation

In order to evaluate the performance, a trace-driven simulation is performed on the board. Through the UART, the I/O request is transmitted from the trace to the board and the response time is measured. The trace was collected on a PC running windows7 operating system while installing programs, surfing the Internet, editing documents, and so on. Since the partition where the window is installed is very large compared to the storage capacity of the experimental board, the unused hole is removed to adjust the total capacity of the trace to 4 GB. Table 2 shows the characteristics of the modified traces.

Table 2: Trace Attributes

Total read bytes	282 GB
Total written bytes	371 GB

Fig. 3 shows the average performance measurement result. The X axis represents the length of the I/O queue and the Y axis represents the average response time in second. The result shows that the performance difference between the single-threaded FTL and the multi-threaded FTL is negligible until the queue depth is four. However, as the depth of the queue increases, the performance of the multi-thread improves greatly. When the queue length is 256, the average response time is shortened up to about 5.5%.

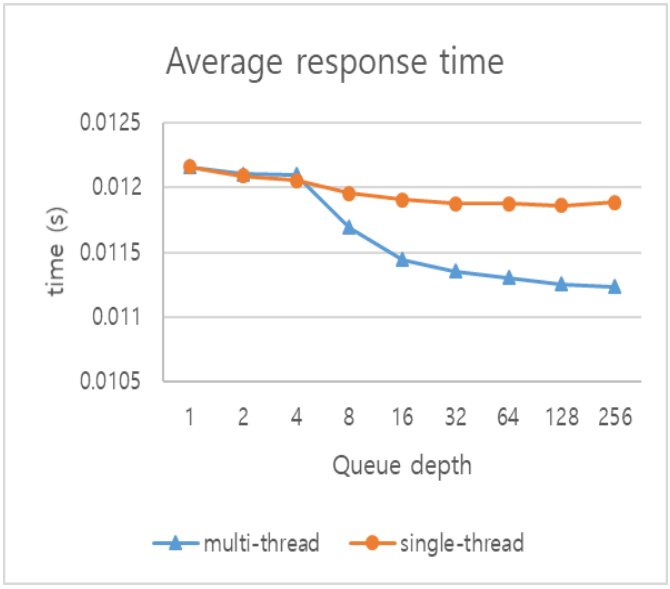

Figure 3: Average response time

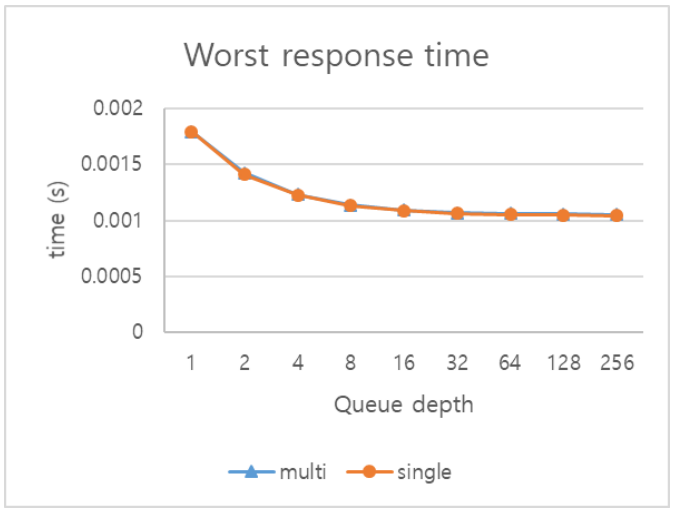

Figure 4: Worst response time

Fig. 4 shows the result of the worst performance measurement of I/O requests. The X axis is the depth of the queue and the Y axis is the worst response time in second. The worst performance improves as the queue length increases, but the difference between the single-threaded FTL and the multi-threaded FTL is insignificant. That is, using the multi-threaded FTL was not effective in improving the worst performance.

CONCLUSION

In this study, we implemented the multi-threaded FTL using uC/OS-II operating system and evaluated its performance on the board equipped with NAND flash memory. The performance evaluation results showed that the multi-threaded FTL improved the average performance up to about 5.5%. However, the worst-case performance was not improved. The limitation of this study is that the experiments were performed on the board with very limited storage capacity and two NAND flash chips. In reality, commercial SSDs provide a large storage capacity and connect multiple NAND flash chips in parallel using multiple channels. We expect that the performance improvement of the multi-thread FTL can be larger in the highly parallelized SSDs.

ACKNOWLEDGEMENTS

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2016R1D1A1A09918559).

REFERENCES

[1] J. Kim, Y. A. Winata, and I. Shin. "Multi-thread Flash Translation Layer for Multi-core Solid State Drives,"

International Journal of Applied Engineering Research, vol. 11, no. 2, pp. 1187-1191, 2016.

[2] A. Ban "Flash file system," United States Patent. No. 5,404,485, April 1995.

[3] A. Ban, "Flash file system optimized for page-mode flash technologies", United States Patent, no. 5,937,425, 1999.

[4] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems", IEEE Trans. Consumer Electron., vol. 48, no. 2, pp. 366–375, 2002.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy "Design tradeoffs for SSD performance," USENIX Annual Technical Conference, pp. 57-70, June 2008.

[6] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in Proc. IEEE HPCA, pp. 266–277, 2011.

[7] Y. A. Winata, K. Sanghoon, and I. Shin "Enhancing internal parallelism of solid-state drives while balancing write loads across dies," Electronics Letters, vol. 51, no. 24, pp. 1978-1980, November 2015.

[8] The OpenSSD Project, http://www.openssd-project.org/wiki/The_OpenSSD_Project