Address Pattern Recognition Flash Translation Layer for Quadruple-level cell NAND-based Smart Devices

Se Jin Kwon[1]
Department of AI Software
Kangwon National University, Samcheok, South Korea

***Abstract*—The price of the solid-state drives has become a major factor in the development of flash memory technology. Major semiconductor companies are developing quadruple-level cell NAND-based SSDs for smart devices. Unfortunately, SSDs composed of quadruple-level cell (QLC) flash memory may suffer from low performance. In addition, few studies on internal page buffering mechanisms have been conducted. As a solution to these problems, an address pattern recognition flash translation layer (APR-FTL) is proposed in this study. APRA-FTL gathers the data in a page unit and separates random data from sequential data. Furthermore, APRA-FTL proposes address mapping algorithm which is compatible to the page buffering algorithm. Experimental results show that APRA-FTL generates a lower number of write and erase operations compared to previous FTL algorithms.**

Keywords—Memory management; nonvolatile memory; smart devices

I. Introduction

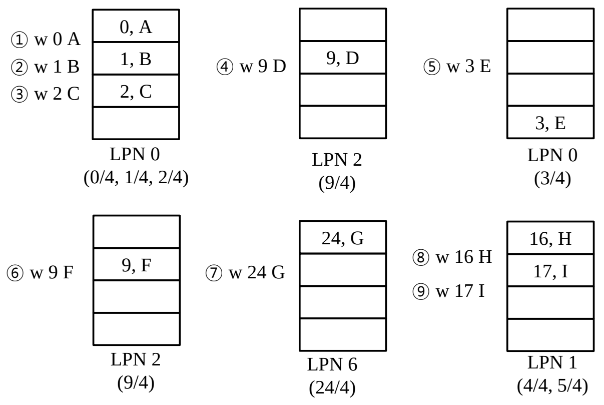
Price has become an important factor in the development of flash memory system, as many semiconductor manufacturing companies are competing for dominance in the smart device market [1]. Some semiconductor companies have recently turned their attention to developing QLC flash memories for smart devices as a means of providing large capacity at a low price [2]. As a result, instead of applying high-performing single-level cell (SLC) or multi-level cell (MLC) technology to smart devices, major semiconductor companies are developing QLC flash memory for smart devices. However, implementing QLC flash memories on smart devices may drastically diminish device performance and durability and even generate inconsistent response times, as smart devices must generate frequent updates from temporary files and metadata [3].

Furthermore, smart devices may execute unnecessary write operations on QLC-based SSDs because of the large page size of QLC flash memory. Although the page size of SLC or MLC flash memory is only four to eight times larger than the data sector of the file system [4], the page size of QLC flash memory is predicted to be 64 to 512 sectors. Because of the large page size of QLC flash memory, there is a considerable chance that the file system may frequently command a page re-access in the flash memory. However, the number of partial programming (NOP) requirements within a page is limited to only one to avoid program-disturbing errors [5]. Therefore, QLC flash memory tends to use an internal register or page buffer to gather data in a page unit.

Well-optimized flash translation layers (FTLs) are based on SLC/MLC/TLC flash memory [6], and therefore they do not give considerable attention to their own page buffering mechanisms (PBMs). This is an issue that has yet to be fully researched. In this study, the implementation of an address pattern recognition flash translation layer is proposed. APRA-FTL gathers the data sectors and rearranges them into the page size of the QLC flash memory, instead of writing data on the flash memory immediately. Furthermore, APRA-FTL proposes address mapping algorithm which is compatible to the page buffering algorithm. The efficient data collection of APRA-FTL provides improved performance and consistent response despite the use of low-performing QLC flash memories.

II. Related Works

Previous PBMs can be classified into fine- and coarse-level PBMs. A fine-level PBM [7] gathers data sectors without a logical address boundary. However, this method reveals every corresponding physical sector number in DRAM. Therefore, it requires approximately 8 GB per 1 TB of flash memory. By contrast, a coarse-level PBM [8][9] gathers data using the same logical page number (LPN). Fig. 1 shows an example of a coarse-level PBM. Here, the extensive data from the write command are broken down into a unit of file system data sectors. A logical sector number (LSN) and its corresponding data are shown as (LSN, data). For simplicity, the size of a block is assumed to be four pages, where each page consists of only four sectors. Because they all belong to LPN 0, (0, A), (1, B), and (2, C) belong to a single page (= 0/4, = 1/4, = 2/4). However, when (16, D) occurs, the data within the page buffer are sealed as a page and written to flash memory because (16, D) belongs to LPN 4 (= 16/4). Finally, the page buffer is flushed, and data "D" are written to the emptied page buffer. Similarly, other data are gathered as a page unit. The coarse-level PBM generates six subpages, as shown in Fig. 1, although the file system issues only nine data sectors. If each page is filled with data, the coarse-level PBM generates only three pages. In Fig. 1, it is assumed that a page consists of only four subpages. However, we should note that the page size of QLC flash memory ranges from 64 to 512 sectors. There is a considerable chance that the space utilization of a QLC-NAND SSD will decrease drastically because of the frequent occurrence of subpages.



total number of page writes: 5, current data in buffer: H, I

Fig. 1. Coarse-level Page Buffering Mechanism.

III. Address Pattern Recognition Flash Translation Layer

A. Address Pattern Recognition Algorithm

Address pattern recognition algorithm (APRA) modifies previous adaptation layer [10] and provides compatible address mapping algorithm according to the experiments in Section III.B. In other words, the address pattern recognition FTL is composed of address pattern recognition algorithm and address mapping algorithm.

The QLC flash memory cannot access to a page for an additional write operation because of the restricted NOP. Therefore, the QLC flash memory requires an efficient PBM. Unfortunately, the coarse-level PBM generates five page write operations, as previously explained in Section II. We should note that the QLC flash memory consists of 8 to 16 sectors per page. Because the amount of random data tends to fill the space of only one to four sectors, it is likely that the scenario will frequently occur. As a solution for the unnecessary write operations, the address pattern recognition algorithm (APRA) has been proposed [10].

APRA is shown as Algorithm I. The main objective of APRA is to segregate the random data from the sequential data and to allow the address mapping algorithm in FTL to separately manage them [10]. When the file system issues a write command along with the LSN, APRA first checks whether (LSN, data) is sequential to the sequential buffer (Algorithm 1 line 1). If (LSN, data) is sequential to the sequential page buffer, it is considered as the sequential data, and therefore it is inserted into the sequential page buffer (Algorithm 1 line 2). On the other hand, if the incoming data's LSN already exists in the page buffers (Algorithm 1 line 3), APRA consider (LSN, data) as an update, and therefore it is inserted into the random page buffer (Algorithm 1 line 4). Finally (LSN, data) is included in the undefined buffer if (LSN, data) is not sequential pattern nor an update.

ALGORITHM I: Address Pattern Recognition

```
Input:  logical sector number (LSN), data (data)
Procedure:  write_page_buffer (LSN, data)
 1:  if (LSN, data) is sequential then
 2:    data_insert(sequential_buffer, LSN, data);
 3:  else if (LSN, data) is update then
 4:    data_insert(random_buffer, LSN, data);
 5:  else
 6:    data_insert(undefined_buffer, LSN, data);
 13  end if

Input:  page buffer (PB), LSN, data
Procedure:  data_insert (PB, LSN, data)
 14:  if PB is full then
 15:    Write PB to flash memory;
 16:    Flush PB;
 17:    Insert (LSN, data) into PB;
 18:  else
 19:    Insert (LSN, data) to PB;
 20:  end if
```

B. Address Mapping Algorithm

The page buffering layer reorganizes the data sectors at the page unit, and the address mapping layer determines the physical address of the data to be written onto the flash memory. To evaluate the performance of the PBM and APRA, previous log-block algorithms were implemented in the address mapping layer.

Log blocks are temporary buffers for physical blocks. Of previous address mapping algorithms, log-block algorithms are well known to be the most optimized FTL algorithms with respect to their DRAM requirements and performance. Previous log-block algorithms made use of variations in the associativity between the blocks and log blocks, which can be classified into block-level associativity, full associativity, and superblock-level associativity. In this study, address mapping algorithms based on block-level associativity, full associativity, and superblock-level associativity are referred to as BAST, FAST, and SAST, respectively.

IV. Performance Evaluation

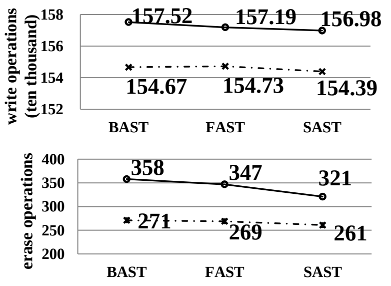
A. Experimental Setup

For this study's experiment, an FTL simulator was developed, and trace-driven simulations were conducted. The traces were retrieved from smartphones running various multimedia services and applications. The developed FTL simulator consisted of two layers: a page buffering layer and an address mapping layer. The previous PBM and APRA were implemented in the page buffering layer, and the number of pages generated by both algorithms were then monitored. It is also assumed that each page and block consist of 64 sectors and 256 pages, respectively, and that the performances of the read, write, and erase operations are 100 µs, 1,500 µs, and 6 ms, respectively. Furthermore, ARPA-FTL is analyzed by implementing block-level, full, superblock-level associative address mapping algorithms.

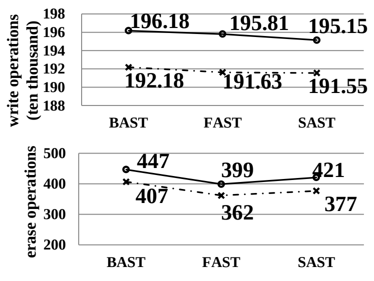
B. Performance Analysis

After PBM and APRA generates the data in the unit of a page, the address mapping layer determines the physical address within the flash memory. Fig. 2 shows the number of write and erase operations executed on BAST, FAST, and SAST. In the case of Trace A, the PBM generated approximately 1,575,200, 1,571,900, and 1,569,800 write and 358, 347, and 321 erase operations in BAST, FAST, and SAST, respectively. By contrast, APRA-FTL it executed approximately 1,546,700, 1,547,300, and 1,543,900 write and 271, 269, and 261 erase operations in BAST, FAST, and SAST, respectively. Similarly, APRA-FTL executed fewer write and erase operations in the overall traces, as shown in Fig. 2.

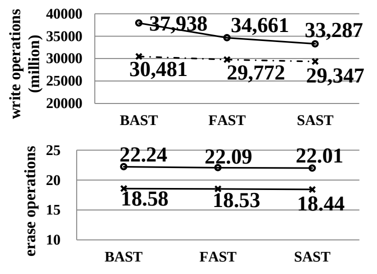
As previously indicated, the number of pages transferred from the page buffering layer considerably affects the number of write and erase operations. A close observation reveals that APRA-FTL avoids numerous operations in the address mapping layer because it collects small-sized random data at the page unit. We should note that the number of write operations executed on the flash memory is higher than that of the pages themselves.



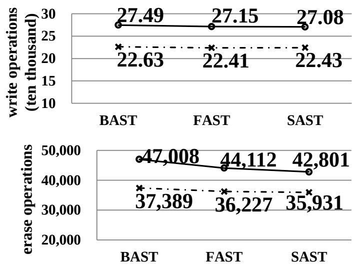
(a) Trace A



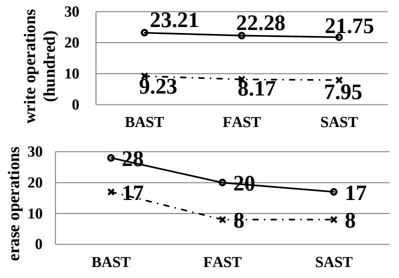
(b) Trace B



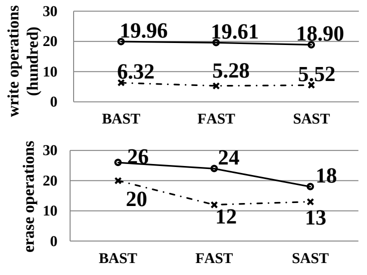
(c) Trace C



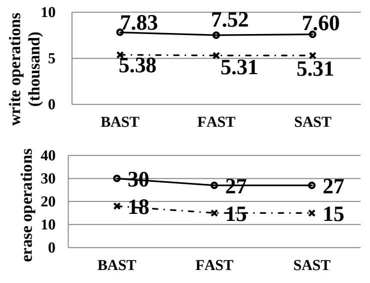
(d) Trace D



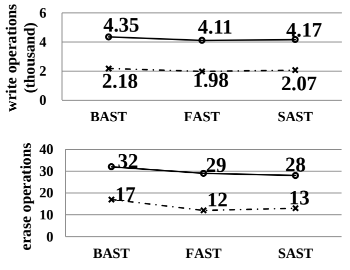
(e) Trace E



(f) Trace F



(g) Trace G



(h) Trace H

Fig. 2. Number of Write and Erase Operations.

The cause of the deviation between the number of pages and write operations might be of interest, as the page buffering layer issues a write operation to the address mapping layer whenever it gathers the data into a page unit. This is because the address mapping layer executes additional write operations internally to copy the data from an old to a new block. When a block is determined to be erased in the address mapping layer, BAST, FAST, and SAST copy the valid data from the old block to a new block and then execute an erase operation on the old data block. Therefore, the number of additional write operations for copying the data is considerably influenced by the number of erase operations. Because APRA-FTL reduces the number of erase operations by gathering random data and avoiding unnecessary write commands, it reduces the overall number of write operations as well.

V. Conclusion

Major semiconductor companies are using QLC flash memory in smart devices to lower the prices of smart devices. Unfortunately, previous page buffering algorithms do not segregate random data, thus generating many unnecessary write commands in the QLC flash memory. This results in a drastic performance degradation in smart devices. Furthermore, there has not been any experiment that considers the compatibility between page buffering algorithm and address mapping algorithm. As a solution, an APRA-FTL was proposed in this study. The APRA-FTL may require additional DRAM or the use of an internal buffer. However, we showed that it accurately identifies random data and thus considerably reduces the number of write operations. From experiments conducted in this study, APRA-FTL reduced the overall number of operations. In a future study, APRA-FTL will be implemented in various wear-leveling algorithms, and additional experiments will be conducted on the durability and power-off recovery of smart devices.

References

[1] MICRON Electronics, “Cache Programming Operations,” MICRON Electronics Technical Notes, 2022.

[2] S. Kumar, P. K. Singh, S. Gupta, “A Survey of Erase Operation in NAND Flash Memory,” 2022 1st International Conference on Informatics (ICI), Noida, India, 2022, pp. 186-190.

[3] Z. Du et al., “A Novel Program Suspend Scheme for Improving the Reliability of 3D NAND Flash Memory,” IEEE Journal of the Electron Devices Society, vol. 10, pp. 98-103, 2022.

[4] Y. Luo, M. Lin, Y. Pan and Z. Xu, “Dual Locality-Based Flash Translation Layer for NAND Flash-Based Consumer Electronics,” IEEE Transactions on Consumer Electronics, vol. 68, no. 3, pp. 281-290, 2022.

[5] W. Zhou et al., “Temporal Correlation Detection Based on 3D NAND Flash In-Memory Computing,” IEEE Electron Device Letters, vol. 43, no. 6, pp. 874-877, 2022.

[6] H. Chen, Y. Dang, H. Wang, X. Xu, J. Zhang and Y. Huang, “Process Optimization and Yield Improvement for Erase Failure in Embedded Flash Memory,” 2022 China Semiconductor Technology International Conference (CSTIC), 2022, pp. 1-3.

[7] P. Jin, C. Yang, X. Wang, L. Yue and D. Zhang, “SAL-Hashing: A Self-Adaptive Linear Hashing Index for SSDs,” IEEE Transactions on Knowledge and Data Engineering, vol. 32, no. 3, pp. 519-532, 2020.

[8] R. Mativenga, J.-Y. Paik, J. Lee, T. S. Chung, and Y. Kim, “RFTL: Improving performance of selective caching-based page-level FTL through replication,” Cluster Comput., vol. 22, no. 1, pp. 1–17, 2019.

[9] D. Lee, D. Hong, W. Choi and J. Kim, “MQSim-E: An Enterprise SSD Simulator,” IEEE Computer Architecture Letters, vol. 21, no. 1, pp. 13-16, 2022.

[10] S. J. Kwon, “An Adaptation Layer for Hardware Restrictions of Quadruple-Level Cell Flash Memories” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 13, no. 8, 2022.