IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 25, 2019, accepted May 28, 2019, date of publication May 31, 2019, date of current version June 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2920249

Address Translation Layer for Byte-Addressable Non-Volatile Memory-Based Solid State Drives

SE JIN KWON, (Member, IEEE)
Department of Computer Engineering, Kangwon National University, Samcheok 25913, South Korea
e-mail: sjkwon@kangwon.ac.kr

This work was supported by the Basic Science Research through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2017R1D1A3B04031440.

ABSTRACT Because of the need for processing and managing the massive amounts of big data in smart/wearable devices and driverless vehicles, semiconductor companies are focusing on developing byte-addressable non-volatile memory (NVM)-based storage systems. The byte-addressable NVMs, such as phase-change memory, resistive memory, and magnetoresistive memory, are regarded as an alternative to NAND flash memories. There have been many proposals and studies on the use of NVM as main memory in the memory hierarchy. However, there has not been much academic research on using NVM as a substitute for NAND flash memories. This paper provides a system architecture for an NVM-based solid state drive based on some speculations/assumptions on the hardware characteristics of NVMs. It applies the previously proposed address-mapping algorithms of conventional solid state drives to the NVM-based solid state drives and examines their suitability. The optimization of I/O parallelism of static and dynamic address mapping algorithms is compared and analyzed. This paper also observes the effect of log block policies on the hardware characteristics of the NVMs.

INDEX TERMS Cache storage, flash memory, nonvolatile memory, software systems, wearable computers.

I. INTRODUCTION

Recently, many semiconductor companies have begun focusing on the ways to process and manage the massive amounts of big data in smart/wearable devices and driverless vehicles. Such devices and machines require high-performance storage systems. As result, the major semiconductor vendors are developing byte-addressable non-volatile memory (NVM) based storage systems, as an alternative to DRAM and NAND flash memories. The byte-addressable NVMs, such as phase-change memory, resistive memory, and magnetoresistive memory, promise to revolutionize I/O performance. Many semiconductor companies predict that byte-addressable NVM will be used in smart phones, wearable devices, and Internet-of-Things devices, because they provide faster read and write performance and longer durability than NAND flash memory. As a result, many researchers are attempting to determine where to place byte-addressable NVMs in the memory hierarchy.

To address the limited scalability of DRAM, there have been many proposals placing NVM on the processor's memory bus alongside the conventional DRAM, leading to hybrid volatile/non-volatile main memory systems [1]. In contrast, there have been no academic proposals, studies, nor mass-produced devices using byte-addressable NVMs as a secondary device. There have been some speculations on 3D Xpoint [2], which reveals a glimpse of new high-performance secondary storage. However, the materials and techniques used in the products remain unknown [3]. As a result, this paper explores the possibility of developing a solid state drive (SSD) based on byte-addressable NVM (BNVM-SSD) instead of NAND flash memory for secondary storage. This paper provides a system architecture for a BNVM-SSD based on some speculations/assumptions on byte addressable NVMs. Furthermore, to provide efficient data transfer between the file system and BNVM-SSD, the BNVM-SSD requires a device driver. Fortunately, there are many flash translation layer (FTL) algorithms based on NAND flash memory. Therefore, this paper evaluates existing NAND-based address mapping algorithms with respect to the I/O parallelism of BNVM-SSD, and finally analyzes the efficiency of the address mapping algorithm with respect to the performance of the BNVM-SSD.

The associate editor coordinating the review of this manuscript and approving it for publication was Zonghua Gu.

TABLE 1. Acronyms definitions.

Non-volatile memory	NVM
Solid state drive	SSD
Solid state drive based on byte-addressable NVM	BNVM-SSD
Flash translation layer	FTL
Out-of-band	OOB
Phase change memory	PCM

II. RELATED WORK

A. STRIPING TECHNIQUES

Unlike other storage devices, NAND flash memory has an erase-before-write characteristic [4]. In contrast, BNVMs do not require a costly erase operation before updating data [5]. Although the hardware characteristics of NAND flash memories and BNVMs are very different, both memories need to fully utilize I/O parallelism to increase the overall performance. As a result, this section reviews the previous address mapping algorithms for NAND flash memory and analyzes the advantages and disadvantages of implementing them on a BNVM-SSD.

The occurrence of I/O parallelism depends upon the address mapping algorithm of the FTL, because the address mapping algorithm determines ''where'' to write the incoming data. An address mapping algorithm adopts either a static or dynamic striping technique for I/O parallelism [6]. A static striping technique maps logical addresses to physical addresses beforehand. Such a technique predetermines the corresponding physical addresses according to the physical offsets of the channel/chips/plane/die/block. A dynamic striping technique, in contrast, allocates logical addresses to physical addresses on any ''idle'' chip, regardless of the physical offsets.

Hu *et al.* [6] and Chen *et al.* [7] compared the average response time of a static striping technique to that of a dynamic striping technique in NAND flash memory. Their experimental results indicate that the response time of dynamic striping outperforms that of static striping for various workloads, because multi-channel SSDs cannot access ''busy'' chips. As a result, a static striping technique may delay the response time when a write request happens to be on a busy chip. In contrast, a dynamic striping technique can avoid such delay by adaptively distributing the write requests to the idle chips. However, it suffers from the need for a large mapping table, because it records all the logical-to-physical addresses in DRAM or NAND flash memory.

B. NAND-BASED ADDRESS MAPPING ALGORITHMS

1) PAGE-LEVEL MAPPING SCHEME

The page-level mapping scheme consists of the algorithms that maintain the logical-to-physical address mapping table in a read/write unit. It can employ both static and dynamic striping techniques. A static page-level mapping scheme predetermines the logical-to-physical addresses. It does not need to maintain a mapping table in DRAM or in the flash memory array, because the mapping information does not change in the static striping technique. In contrast, the dynamic page-level mapping scheme must indicate the physical page numbers corresponding to the logical page numbers (LPNs) [8], because the mapping information can be freely allocated and changed in the dynamic striping technique. The dynamic page-level mapping scheme requires a total of 64 MB per 128 GB of a triple-level cell SSD. Under the assumption that BNVM-SSDs contain a controller similar to those in NAND-SSDs, the mapping table size is determined by multiplying the total number of pages by the mapping information size.

Because the page-level mapping table for the dynamic striping technique is too large to upload onto the internal DRAM [9], there have been some proposed methods that store the page-level mapping information in the flash memory array. Qin *et al.* [10] suggested managing the page-level mapping information in the out-of-band (OOB) area of each page. According to their experiments, MNFTL [10] is capable of storing the page-level mapping information of a block within the OOB area of two physical pages. As a result, MNFTL reads the two latest physical pages within a block to inquire about the page-level mapping information of the corresponding physical block. Likewise, Gupta *et al.* [11] dedicated a few blocks for storing page-level mapping information.

As shown above, a dynamic page-level mapping scheme requires additional read/write operations for retrieving the mapping information from the flash memory array. As a result, a dynamic page-level mapping scheme may reduce the number of erase operations by changing the mapping information compared to that of the static page-level mapping scheme; however, a static page-level mapping scheme may perform better in sequential read/write patterns [6].

2) BLOCK-LEVEL MAPPING SCHEME

Unlike a page-level mapping scheme, a block-level mapping scheme cannot apply dynamic striping at page-level, because it stores only physical block numbers (PBNs) in its mapping table [12]. As a result, block-level mapping information can be freely altered, but the data must be read/written according to the offset within a block. There are 216 blocks in 128 GB of triple-level-cell based SSD, and each block-level mapping entry requires 3 bytes. As a result, a block-level mapping table requires only 0.19 MB.

The mapping table size of the block-level mapping scheme is very small compared to that of the page-level mapping scheme. However, the block-level mapping scheme cannot stripe the sequential data at channel- or chip-level, because it must write the data according to the offset within a block. For instance, assume that the file system issues the write request ''w 0 $A_{0\text{-}3}$, 4,'' which says to write data A of size 4 starting from LPN 0. Moreover, assume the physical block corresponding to logical block number (LBN) 0 (LPN 0–LPN 128) exists in chip 0. In this case, four write

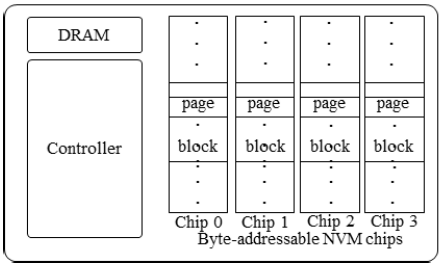


FIGURE 1. Schematic structure of BNVM-SSD.

operations corresponding to LPN 0, LPN 1, LPN 2, and LPN 3 are issued to chip 0, even though chips 1, 2, and 3 could be idle. As a result, the total response time for "w 0 $A_{0\text{-}3}$" is four write operations in the block-level mapping scheme. Note that, in the same scenario, if all chips are in the idle state, a static/dynamic page-level mapping scheme requires only one write response time for writing A_0, A_1, A_2, and A_3 on chips 0, 1, 2, and 3, respectively.

3) LOG-BLOCK POLICIES

Yao *et al.* [13], Kim *et al.* [14], Park *et al.* [15], Jung *et al.* [16], and Shim *et al.* [17] proposed adding log-block policies to the block-level mapping scheme. These log-block policies allocate a fixed percentage of log blocks as a temporary buffer for updates and share them with the entire SSD. The policies allocate 0.1% of flash memory for log blocks and maintain a page-level mapping table for the log blocks in DRAM. Such log-block policies differ with respect to how the log blocks are allocated to the data blocks. Qin *et al.* [10], Yao *et al.* [13], and Kim *et al.* [14] proposed dedicating one log block per data block. When an update occurs on a data block, they dedicate one log block for the corresponding data block and write the update to that log block. Because it dedicates one log block for one data block, this log block policy is referred to as "coarse associativity" in this study. In contrast, Shim *et al.* [17] proposed "full associativity," which allows the sharing of all log blocks for all the data blocks.

III. BYTE-ADDRESSABLE NON-VOLATILE SSD

A. HARDWARE ARCHITECTURE

Figure 1 overviews the hardware architecture and characteristics of BNVM-SSD. There have not been many academic proposals/researches on using NVMs as a secondary storage, and there is not yet a commercial BNVM-SSD product. As a result, for the purposes of this study, the hardware architecture of BNVM-SSD was designed based on the following assumptions.

We first assume that the BNVM-SSD consists of a controller, an internal DRAM/SRAM, and NVM chips. The BNVM-SSD uses an internal DRAM/SRAM under the assumption that the read/write speed of DRAM/SRAM outperforms that of NVM. When the file system issues the read/write requests, the controller determines "where" to write the data. We also assume that the controller contains a software layer called the non-volatile translation layer (NTL), which plays a role very similar to that of NAND flash memory's FTL. The main role of NTL is to convert the given logical addresses to physical addresses of the NVM array according to its address mapping algorithm. The controller maintains the logical-to-physical address mapping table in the internal DRAM/SRAM.

A BNVM-SSD is assumed to be composed of multiple chips, and we assume it uses channel/chip-level parallelism, which enables each chip to read/write the incoming data independently. Furthermore, it does not require the units of bank, plane, or block, because it does not require an erase operation, as NAND-SSDs do. Finally, this study assumes that the BNVM-SSD is read or written using the units of a sector or page. Yue and Zhu [18] and Mittal *et al.* [19] proposed the Parallel Chips Phase Change Memory (PCM), which augments a PCM bank with extra chips to increase the number of bits that can be written to the bank in parallel. According to their experiments, the increased write unit size reduces the number of write units and hence shortens the time required to complete writing a cache line to PCM [20]. This paper applies the same methodology to the BNVM-SSD. To remain compatible with current file systems, the units of write/read are sectors (512 B) or pages (2 KB or 4KB).

B. HARDWARE CHARACTERISTICS

We assume that the BNVM-SSD is composed of multiple phase change memory (PCM) chips. However, note that other NVMs, such as resistive memory and magnetoresistive memory, have the potential to be implemented in a BNVM-SSD, and the proposed approach in this study is not limited to PCM chips [21]. The hardware characteristics of PCM are compared with those of NAND flash memory, to evaluate the implementation of existing NAND-based mapping algorithms for BNVM-SSD in Section IV.

The most distinct difference between PCM and NAND is the overwrite operation. Unlike NAND flash memory, PCM is capable of updating data without a preceding erase operation. However, the cost of an update is larger than that of writing on an empty space [22]. In this study, any write operation executed on an empty page is referred to as a "clean write" and an update is referred to as a "dirty write." Although PCM does not have an erase operation as in NAND, it is supported by a refresh operation, which cleans a fixed number of read/write units. A PCM array is divided into refresh units, and each refresh unit is composed of a fixed number of read/write units. For convenience, the terminology used for NAND-SSD is re-used for the BNVM-SSD. Hence, read/write and refresh units are defined as "pages" and "blocks," respectively.

Once a page is occupied with data, PCM cannot execute a clean write unless a refresh operation precedes it on the corresponding block. Otherwise, PCM must execute a dirty write on the page. However, note that the cost of a refresh operation (e.g., 50–100 ms) is much higher than that of a clean (200 μs) or dirty write (250 μs) [22]. As a result, NTL must carefully consider the usage of the refresh operation

to avoid degrading performance. In addition to performance, each cell in PCM has a limited life cycle for write/refresh operations. If a page has been written or refreshed over a certain threshold (e.g., 1,000,000 times), the corresponding page may not function correctly and is thus unable to guarantee data integrity [22].

IV. PERFORMANCE EVALUATION

The following subsections describes the application of existing address mapping algorithms to a BNVM-SSD and observes their effect on its performance.

A. PAGE-LEVEL AND BLOCK-LEVEL MAPPING SCHEMES

1) COMPARISON

A BNVM-SSD is assumed to be composed of multiple chips and use channel-/chip-level parallelism, which enables each chip to read or write the incoming data independently. Therefore, when a page-level mapping scheme is implemented on a BNVM-SSD, it is capable of writing sequential data on idle chips simultaneously. Let us assume that the file system issues the command "w 0 $A_{0\text{-}3}$, 4." If chips 0, 1, 2, and 3 are in the idle state, the page-level mapping scheme can write data $A_{0\text{-}3}$ in one write response time by simultaneously issuing the write commands corresponding to A_0, A_1, A_2, and A_3 on chips 0, 1, 2, and 3, respectively. Of course, one write response time is only one possibility; the actual write response time will differ depending upon the static and dynamic striping techniques used.

As shown above, the page-level mapping scheme can increase I/O parallelism using static or dynamic striping techniques. In a BNVM-SSD, channel-/chip-level parallelism is a very important factor for increasing data throughput, because the BNVM-SSD does not have any other units, such as banks, planes, or blocks, as in NAND-SSD (according to the assumptions in Section III). Unlike the page-level mapping scheme, the block-level mapping scheme maps the logical-to-physical addresses in block units. It cannot stripe the sequential data at channel- or chip-level, because it bounds the logical addresses to their corresponding blocks. Assume that the file system issues the write request "w 0 $A_{0\text{-}3}$, 4." The block-level mapping scheme first searches for the corresponding physical block in its mapping table. Here, we assume that the physical block mapped to LBN 0 (LPN 0 to LPN 127) is PBN 0 and PBN 0 exists in chip 0. Because the block-level mapping scheme dedicates PBN 0 for LBN 0, it must access PBN 0 to write data A. As a result, four consecutive write commands corresponding to A_0, A_1, A_2, and A_3 are issued to chip 0 without considering other idle chips. Because chip 0 cannot initiate the next write command until the previous write command is finished, the block-level mapping scheme consumes four write response times to perform "w 0 $A_{0\text{-}3}$, 4."

2) ANALYSIS

As shown in the above example, the block-level mapping scheme delays the write response time in the BNVM-SSD, because it cannot process the sequential data in the BNVM-SSD simultaneously. Here, the sequential data refers to user data in which the logical addresses are sequential and the data size is large. Unfortunately, real-life workloads, especially multimedia data patterns, are mainly composed of sequential data, although small sized random data may frequently appear as an update. Because the block-level mapping scheme cannot read/write sequential data simultaneously, the block-level mapping scheme is omitted in the following experiments for the sake of brevity. However, the log block policy mentioned in Section II.C is reviewed and applied to the page-level mapping scheme to consider the frequent updates of random data.

TABLE 2. Experimental setup.

Clean write (page)	200 µs
Dirty write (page)	250 µs
Refresh (block)	50-100 ms
Cell write cycle limit	1,000,000
Chip parallelism	4
Traces	SNIA IOTTA

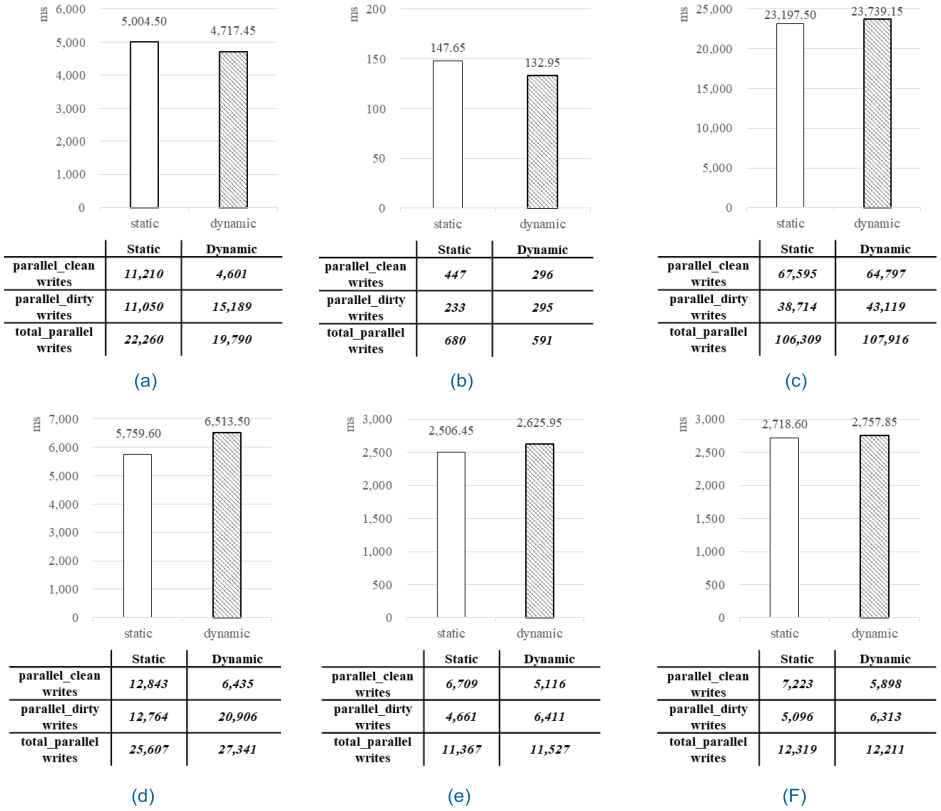
B. ANALYSIS OF STATIC AND DYNAMIC MAPPING SCHEMES

1) COMPARISON

The main goal of this subsection is to analyze the difference in performance between pure static and dynamic page-level algorithms in a BNVM-SSD. Readers may question the reason for this comparison, because the dynamic page-level mapping algorithm suffers from the need for a large mapping table in a BNVM-SSD. However, the dynamic page-level mapping algorithm is well-known for its optimized I/O parallelism in NAND-SSDs. As a result, it is worthwhile considering how compatible the static and dynamic page-level mapping algorithms are with the hardware architecture and characteristics of the BNVM-SSD.

The BNVM-SSD can overwrite data using a dirty write operation or a refresh operation. The refresh operation cleans a fixed number of read/write units, and therefore, it requires a mechanism that collects the updates in a refresh unit. Unfortunately, neither pure static nor dynamic page-level mapping algorithms have such a mechanism. Furthermore, the cost of a dirty write operation (250μs) is much smaller than that of a refresh operation (e.g., 50–100 ms) [22], and a refresh operation generates additional clean write operations for copying the valid data to other areas. As a result, a dirty write operation is used for the updates in the comparison between the pure static and dynamic page-level mapping algorithms.

The experimental setup is given in Table 2. The simulation mimics the hardware characteristics of BNVM-SSD based on the assumptions in Section III and runs various traces from SNIA IOTTA [23] to ensure the experimental results are reproducible. The cost of a clean write, dirty write, and erase is 200 μs, 250 μs, and 100 ms, respectively. Traces A to F are workloads from NEXUS5. Trace A contains write-intensive patterns retrieved from music applications and Twitter.



	Static	Dynamic
parallel_clean writes	11,210	4,601
parallel_dirty writes	11,050	15,189
total_parallel writes	22,260	19,790

	Static	Dynamic
parallel_clean writes	447	296
parallel_dirty writes	233	295
total_parallel writes	680	591

	Static	Dynamic
parallel_clean writes	67,595	64,797
parallel_dirty writes	38,714	43,119
total_parallel writes	106,309	107,916

	Static	Dynamic
parallel_clean writes	12,843	6,435
parallel_dirty writes	12,764	20,906
total_parallel writes	25,607	27,341

	Static	Dynamic
parallel_clean writes	6,709	5,116
parallel_dirty writes	4,661	6,411
total_parallel writes	11,367	11,527

	Static	Dynamic
parallel_clean writes	7,223	5,898
parallel_dirty writes	5,096	6,313
total_parallel writes	12,319	12,211

FIGURE 2. Comparison on static and dynamic page-level mapping schemes. (a) Trace A. (b) Trace B. (c) Trace C. (d) Trace D. (e) Trace E. (F) Trace F.

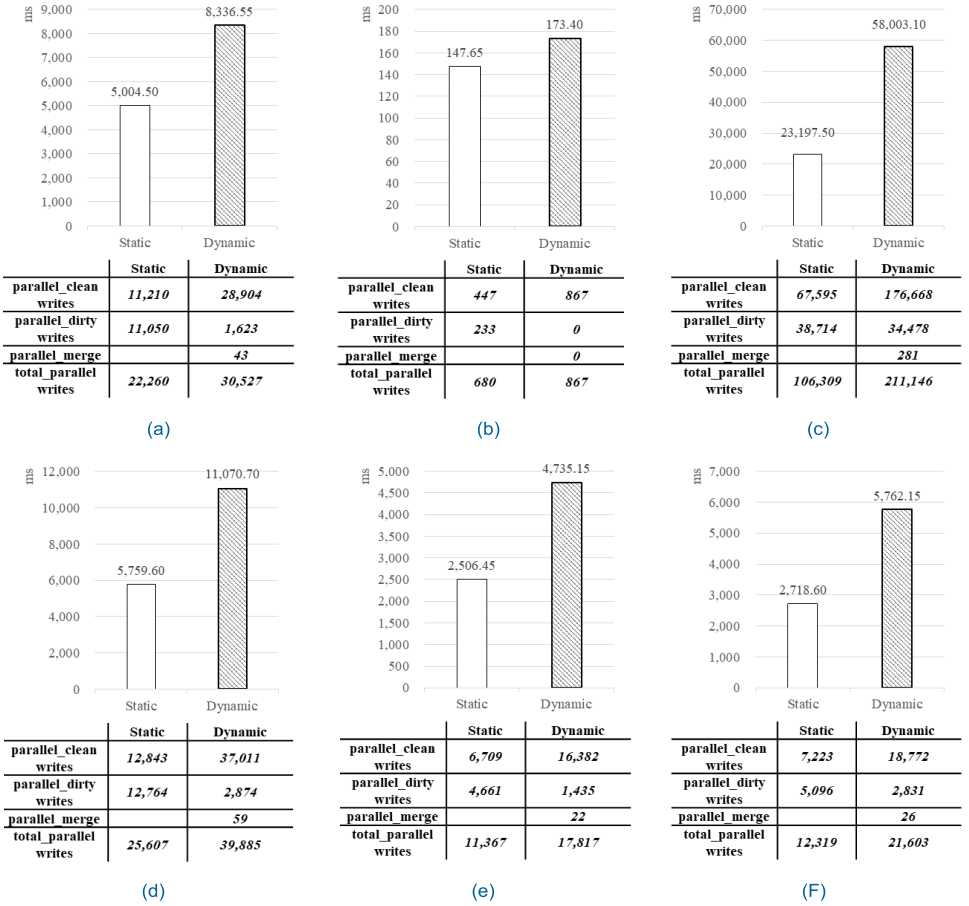
Trace B contains write commands generated by executing multiple copy operations. Traces C, D, and E are composed of read and write commands retrieved from OS installation, Facebook apps, and Google Maps, respectively. Finally trace F consists of read and write patterns generated by multiple activities in web browsers.

Figure 2 compares the static and dynamic page-level mapping algorithms on the BNVM-SSD using traces A to F. The dynamic page-level mapping algorithm outperformed the static page-level mapping algorithm for traces A and B by reducing the total number of parallel_writes. In the results, the number of parallel_ dirty_writes increased in trace B. A close observation of trace B reveals that each parallel_dirty_write in the dynamic page-level mapping algorithm contained only one or two dirty write operations, and the rest were clean write operations. In other words, the dynamic page-level mapping algorithm reduced the total number of parallel_writes by simultaneously performing the clean and dirty write operations in one write response time. As a result, the dynamic page-level mapping algorithm reduced the total number of parallel_writes by increasing I/O parallelism while increasing the number of parallel_dirty_writes.

For traces C, D, E, and F, the static page-level mapping algorithm unexpectedly generated better performance than the dynamic page-level mapping algorithm. The static page-level mapping algorithm generated more parallel_clean_writes than the dynamic page-level mapping algorithm, whereas the dynamic page-level mapping algorithm generated more parallel_dirty_writes. For example, for trace C, the static page-level mapping algorithm generated 67,595 parallel_clean_writes and 38,714 parallel_dirty_writes, whereas the dynamic page-level mapping algorithm generated 64,797 parallel_clean_writes and 43,119 parallel_dirty_writes. The dynamic page-level mapping algorithm reduced the number of parallel_clean_writes by 2,798, but generated an additional 4,149 parallel_dirty_writes when compared with the static page-level mapping algorithm. As a result, the dynamic mapping algorithm increased the total number of parallel_writes, and therefore, the static page-level mapping algorithm outperformed the dynamic page-level mapping algorithm for traces C, D, and E. For trace F, the static page-level mapping algorithm performed better than the dynamic page-level mapping algorithm, although the static page-level mapping algorithm generated more total parallel_writes. This is due to the fact that the cost of a dirty write operation (250μs) is higher than that of a clean write operation (200μs).

2) ANALYSIS

As shown above, the performance of dynamic and static algorithms varied depending on the traces. This phenomenon occurs because of the dirty write operation. When a chip is



	Static	Dynamic
parallel_clean writes	11,210	28,904
parallel_dirty writes	11,050	1,623
parallel_merge		43
total_parallel writes	22,260	30,527

	Static	Dynamic
parallel_clean writes	447	867
parallel_dirty writes	233	0
parallel_merge		0
total_parallel writes	680	867

	Static	Dynamic
parallel_clean writes	67,595	176,668
parallel_dirty writes	38,714	34,478
parallel_merge		281
total_parallel writes	106,309	211,146

	Static	Dynamic
parallel_clean writes	12,843	37,011
parallel_dirty writes	12,764	2,874
parallel_merge		59
total_parallel writes	25,607	39,885

	Static	Dynamic
parallel_clean writes	6,709	16,382
parallel_dirty writes	4,661	1,435
parallel_merge		22
total_parallel writes	11,367	17,817

	Static	Dynamic
parallel_clean writes	7,223	18,772
parallel_dirty writes	5,096	2,831
parallel_merge		26
total_parallel writes	12,319	21,603

FIGURE 3. Comparison on static and static + full page-level mapping schemes. (a) Trace A. (b) Trace B. (c) Trace C. (d) Trace D. (e) Trace E. (F) Trace F.

in a busy state, the static page-level mapping algorithm may suffer from a performance delay while waiting for the chip's status to change to idle. In other words, it increases the overall response time because the frequency and duration of the chip occupancy increases. In a NAND-SSD, an erase operation is the main cause of such chip occupancy, because its cost is much higher than that of a write. As a result, the dynamic striping technique outperformed the static striping technique when the trace contained update patterns that frequently triggered erase operations. However, a BNVM-SSD does not have an erase operation. When an update occurs, both static and dynamic page-level mapping algorithms execute a dirty write operation according to its mapping information. There is no practical performance delay for waiting for the corresponding chip to finish a dirty write operation because the response time deviation between a clean (200 μs) and dirty write (250 μs) operation is only 50 μs [22]. As a result, the dynamic page-level mapping algorithm does not benefit from the flexible address mapping allocation in the BNVM-SSD.

C. ANALYSIS ON FULL ASSOCIATIVE LOG POLICY ON BNVM-SSD

Previous log block policies can be categorized into coarse and full associativities, as explained in Section II. However, a BNVM-SSD cannot apply coarse associativity unless it adopts a block-level mapping scheme. Unfortunately, the block-level mapping scheme is not adequate for the BNVM-SSD, as discussed in Section III.A. As a result, full associativity is applied on the static page-level mapping scheme and the effect on the BNVM-SSD is observed. For convenience, the static page-level mapping algorithm with the full associativity log policy is referred to as the static + full algorithm.

1) MODIFICATION

When an update occurs, the static + full algorithm avoids dirty writes by transferring the updates to the log area instead of immediately executing a dirty write operation in the data area. However, if the updates are immediately directed to the log area, the static + full algorithm cannot take advantage of chip-level I/O parallelism, thus increasing the number of parallel_clean_writes. To avoid such a scenario, we allocate a buffer for collecting the updates. The size of the buffer is the page size times the number of chips. When the buffer is fully filled with the updates, the static + full algorithm executes a clean write operation on each chip's log area simultaneously, resulting in one parallel_clean_write.

When the log area is fully filled with the updates, the statica + full algorithm triggers a merge process. A merge

process is a procedure through which the data area is reclaimed when the log area is full. Its response time is given as follows.

$$T_{merge} = T\_dirty\_write \times M\_valid\_pages + T\_refresh \times refresh\_units \quad (1)$$

Here, the total response time for a merge process (T_{merge}) includes the time taken to copy pages corresponding to the valid data from the log to the data area as well as the time taken to refresh the log area. For the time taken to copy the valid pages, the time for one dirty write (T_{dirty_write}) is repeated for the maximum number of valid pages of the refresh units of the log area (M_{valid_pages}). For example, if the log area contains two valid pages belonging to chip 0 and one valid page for chips 1, 2, and 3, the total dirty write response time consists of two dirty writes. The first valid page belonging to chip 0 is concurrently executed with the other valid pages of chips 1, 2, and 3. Then, the second valid page belonging to chip 0 is transferred from the log to the data area. The response time for a merge process includes the time for refreshing the log area, as given in (1). To simultaneously execute the refresh operations, the same number of refresh units is allocated per chip for the log area, and the refresh units are located in the same offset within each chip. By concurrently executing the refresh operations, the total refresh response time is given by repeating the time for one refresh operation ($T_{refresh}$) by the number of refresh units per chip (*refresh_units*).

2) ANALYSIS

The static + full algorithm is compared to the static page-level mapping algorithm in Figure 3. As expected, the static + full algorithm performed fewer parallel_dirty_writes for all traces by collecting the updates and transferring them to the log area. However, despite reducing the number of parallel_dirty_writes, the static + full algorithm suffers from the following problems.

a: REFRESH OPERATION

The full associativity log policy triggers a refresh operation when the log area is fully filled with the data. Unfortunately, a refresh operation costs 50 to 100 ms in PCM and is used for erasing the entire data area. In other words, the controller is not optimized for cleaning a group of pages in the current PCMs [22]. In conclusion, a BNVM-SSD requires i) an improvement in the performance of the refresh operation and ii) to adopt a controller that can clean a group of pages if it is to adopt the full associativity log policy.

b: INCREASE IN THE NUMBER OF PARALLEL_CLEAN_WRITES

The number of parallel_clean_writes drastically increases. The static + full algorithm generates approximately 1.5 to 1.8 times more parallel_clean_writes than the static page-level algorithm, as shown in Figure 3. However, a close observation reveals that this unexpected performance degradation is caused by the frequent occurrence of large updates. The static + full algorithm collects the updates in the buffer in order to write the updates on each chip's log area simultaneously, as explained in this subsection. However, as a side effect of implementing a buffer, the static + full algorithm generates two parallel_clean_writes when the data is transferred from the buffer to the log area. Assume that chips 0 and 1 are busy processing the previous write request and the buffer is fully filled with the data because of the large update. The static + full algorithm accesses all the chip's log area simultaneously to transfer the data from the buffer to the log area. However, in this case, the data in the buffer has to wait for chips 0 and 1 to finish the previous write request, even though the rest of the chips are in the idle state. As a result, the static + full algorithm delays the response time by two parallel_clean_writes whenever the buffer is full.

c: SUDDEN POWER-OFFS

In addition to degraded performance, the static + full algorithm may suffer from a lack of data integrity if there is a sudden power-off. Because it stores the updates in the buffer temporarily before writing them to the log area, the updates can be vulnerable to sudden power-offs. To deal with this possibility, the BNVM-SSD will need to adopt a power-off recovery mechanism, as in the NAND-SSD. However, such a power-off recovery mechanism requires additional accesses to the spare area of each page. As a result, the static + full algorithm may need to increase the number of parallel_clean_writes to implement a power-off recovery mechanism.

V. CONCLUSION

To address the need for high I/O performance, byte-addressable NVMs are attracting a lot of attention from major semiconductor vendors. There have been many proposals that place NVMs on the processor's memory bus alongside DRAM, leading to a hybrid main memory system. However, there has been a lack of academic proposals/studies on using NVMs as a substitute for NAND flash memories. This paper provided a system architecture for the BNVM-SSD, reviewed and applied static and dynamic mapping schemes, and analyzed the effect of the log policy on the BNVM-SSD. According to the experimental results, the performance of the static and dynamic page-level mapping schemes varies depending on the traces. In other words, the dynamic page-level mapping scheme is not able to benefit from I/O parallelism, because the BNVM-SSD does not have a block unit (erase) operation, as in conventional SSDs. Furthermore, the log policy is not compatible with the current the hardware characteristics (assumption/speculations) of the BNVM-SSDs, because the performance cost of a refresh operation is too high compared to that of a write operation. To benefit from the log policy, the refresh operation will need to have smaller units and improve its performance.

In this paper, the performance evaluation was performed by a trace-driven simulation using various traces from SNIA IOTTA. The simulation enables FTL designers to

mimic the hardware characteristics of BNVM-SSD based on the hardware assumptions and speculations, and also enables other researchers to reproduce the experimental results. However, the performance of such secondary storage could also be influenced by the power consumption, material, device size, and operation voltages. In future, as more information on BNVM-SSDs is revealed (e.g., further announcements or products), experiments are planned to evaluate various address mapping algorithms on BNVM-SSD test boards to measure the effect of power, operation voltages, and material on performance. Furthermore, various wear-leveling algorithms will be implemented on BNVM-SSD test boards to determine their effect on durability.

REFERENCES

[1] A. M. Caulfield, A. De, J. Coburn, T. I. Mollov, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Atlanta, GA, USA, Dec. 2010, pp. 385–395.
[2] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, Sep. 2017.
[3] K. Son, K. Cho, S. Kim, G. Park, K. Son, and J. Kim, "Modeling and signal integrity analysis of 3D XPoint memory cells and interconnections with memory size variations during read operation," in *Proc. IEEE Symp. EMC, SI PI*, Long Beach, CA, USA, Jul. 2018, pp. 223–227.
[4] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surv*, vol. 37, no. 2, pp. 138–163, Jun. 2005.
[5] S. He, Y. Wang, Z. Li, X.-H. Sun, and C. Xu, "Cost-aware region-level data placement in multi-tiered parallel I/O systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 1853–1865, Jul. 2017.
[6] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. ICS*, Tucson, AZ, USA, May 2011, pp. 96–107.
[7] F. Chen, B. Hou, and R. Lee, "Internal parallelism of flash memory-based solid-state drives," *ACM Trans. Storage*, vol. 12, no. 3, p. 13, Jun. 2016.
[8] D. Liu, K. Zhong, T. Wang, Y. Wang, Z. Shao, E. H.-M. Sha, and J. Xue."Durable address translation in PCM-based flash storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 475–490, Feb. 2017.
[9] S. J. Kwon, "A cache-based flash translation layer for TLC-based multimedia storage devices," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 1, p. 11, Feb. 2016.
[10] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "MNFTL: An efficient flash translation layer for MLC NAND flash memory storage systems," in *Proc. 48th ACM/EDAC/IEEE DAC*, Jun. 2011, pp. 17–22.
[11] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. ASPLOS*, Washington, DC, USA, Mar. 2009, pp. 229–240.
[12] T. Shinohara, "Flash memory card with block memory address arrangement," U.S. Patent 5 905 993 A, May 18, 1999.
[13] Y. Yao, X. Kong, J. Zhou, X. Xu, W. Feng, and Z. Liu, "An advanced adaptive least recently used buffer management algorithm for SSD," *IEEE Access*, vol. 7, pp. 33494–33505, 2017.
[14] D. Kim, K. H. Park, and C.-H. Youn, "SUPA: A single unified read-write buffer and pattern-change-aware FTL for the high performance of multi-channel SSD," *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–30, Dec. 2017.
[15] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 4, p. 38, Jul. 2008.
[16] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme," *ACM Trans. Embedded Comput. Syst.*, vol. 9, no. 4, p. 40, Mar. 2010.
[17] G. Shim, Y. Park, and K. H. Park, "A hybrid flash translation layer with adaptive merge for SSDs," *ACM Trans. Storage*, vol. 6, no. 4, p. 15, May 2011.
[18] J. Yue and Y. Zhu, "Making write less blocking for read accesses in phase change memory," in *Proc. IEEE 20th Int. Symp. (MASCOTS)*, Washington, DC, USA, Aug. 2012, pp. 269–277.
[19] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 2015.
[20] H.-J. Kim and J.-S. Kim, "A user-space storage I/O framework for NVMe SSDs in mobile smart devices," *IEEE Trans. Consum. Electron.*, vol. 63, no. 1, pp. 28–35, Feb. 2017.
[21] D. H. Kang and Y. I. Eom, "FSLRU: A page cache algorithm for mobile devices with hybrid memory architecture," *IEEE Trans. Consum. Electron.*, vol. 62, no. 2, pp. 136–143, May 2016.
[22] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: A prototype phase change memory storage array," in *Proc. HotStorage*, Jun. 2011, pp. 1–5.
[23] SNIA IOTTA. *SNIA IOTTA Repository Traces*. Accessed: Apr. 1, 2019. [Online]. Available: http://iotta.snia.org

SE JIN KWON (M'16) received the M.S. and Ph.D. degrees in computer engineering from Ajou University, South Korea, in 2008 and 2012, respectively. He was a Research Professor with the Department of Information and Computer Engineering, Ajou University, from 2013 to 2016. He was also a Postdoctoral Researcher with the University of California at Santa Cruz, Santa Cruz, in 2016. He is currently an Assistant Professor with the Department of Computer Engineering, Kangwon National University, South Korea. His current research interests include nonvolatile memory systems, reliable storage systems, and large database systems.