IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 November 2024, accepted 31 December 2024, date of publication 16 January 2025, date of current version 29 January 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3530776

RESEARCH ARTICLE

Don't Cache, Speculate!: Speculative Address Translation for Flash-Based Storage Systems

HYUNGJIN KIM[1,2], SEONGWOOK KIM[3], (Graduate Student Member, IEEE), JUNHYEOK PARK[3], (Graduate Student Member, IEEE), GWANGEUN BYEON[3], (Graduate Student Member, IEEE), AND SEOKIN HONG[3], (Member, IEEE)

[1]Department of Semiconductor and Display Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea
[2]Memory Division, Samsung Electronics, Hwaseong 18448, Republic of Korea
[3]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea

Corresponding author: Seokin Hong (seokin@skku.edu)

This work was supported in part by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korean Government (MSIT) (No.II221170, 25%), in part by the IITP grant funded by the MSIT (No.00228970, 25%), and in part by the Information Technology Research Center (ITRC) support program (No.II212052, 50%).

ABSTRACT Address translation using a logical-to-physical (L2P) mapping table is essential for the NAND Flash-based SSDs. Unfortunately, the L2P mapping table size increases as SSD capacity increases. The mapping table is basically stored in the NAND flash, and a small number of the table entries are cached in the DRAM, leading to performance degradation due to the overhead of loading the mapping table entries from the slow NAND flash. The performance overhead of the address translation is more severe in low-cost flash-based storage systems (e.g., DRAM-less SSD) because they do not employ the DRAM for caching the mapping table, and thus, every I/O request involves an additional read request to the flash to load an address mapping information. To tackle the address translation overhead in the SSDs, this paper proposes *ASTRO* framework that speculatively translates the logical addresses to physical ones by maintaining the contiguity in the address mappings as much as possible. ASTRO consists of three novel mechanisms: 1) Lazy Page Ordering (LPO) to rearrange the pages to maintain the contiguity in the address mappings for each region, 2) Speculative Read (SpecREAD) to convert logical addresses to physical addresses speculatively, and 3) Contiguity Checking (ContCHECK) to monitor the updates in the rearranged regions. These three mechanisms are implemented in the FTL software, and some functions are accelerated by adding simple hardware to the SSD controller. Experimental results demonstrate that ASTRO enhances SSD performance by an average of 80% and 34% for synthetic random read workloads and real-world workloads, respectively, while minimally impacting the write amplification factor.

INDEX TERMS Flash translation layer, address translation, storage system.

I. INTRODUCTION

Solid-state drives (SSDs) have fundamentally transformed data storage paradigms by offering numerous advantages over traditional Hard Disk Drives (HDDs), such as faster access times, lower power consumption, and enhanced durability [1]. In particular, the NAND flash-based SSDs play a crucial role as mainstream storage devices in a broad range of computing systems from smartphones to large-scale data centers [31]. With data characteristics continually growing in complexity and volume, improving the performance and efficiency of the NAND flash-based SSDs has become a critical challenge in computing system design [1], [38].

In SSDs, the Flash Translation Layer (FTL) serves as an intermediary between the operating system and the NAND flash memory [10]. Unlike traditional HDDs where data can be overwritten directly, NAND flash memory requires the erase operation before writing new data to the memory cells. Since the erase operation of the NAND flash is very slow, SSDs typically use an out-of-place update scheme that writes the new data to a free memory region, which is erased in advance, and then invalidates the old data [30]. Due to the

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

out-of-place update scheme, the physical address for specific data can be changed dynamically in the SSDs, requiring the logical-to-physical (L2P) address translation. The FTL is responsible for translating logical addresses to physical ones [10]. For the address translation, the FTL employs an L2P mapping table where each entry maintains the mapping between a logical address from a file system to a physical address of flash memory.

Since the L2P mapping table is large in size, it is typically stored in the flash memory and its entry is fetched for every I/O request, significantly impacting the SSD's performance [10], [15], [19], [22]. To improve the performance of the address translation, many SSDs use some part of a built-in DRAM as a cache memory to store the mapping table [15], [22]. However, this approach tends to incur costs for the additional hardware and increase power consumption. In addition, as SSD capacity increases, the mapping table size also increases, making it impractical to store the whole mapping table in the cache memory. Since the cache memory size is usually much smaller than the mapping table, many data-intensive workloads with random access patterns suffer from frequent misses in the mapping table cache, leading to significant performance degradation. Furthermore, DRAM-less flash-based storage products (e.g., Client SSD [39] and UFS mobile storage [15]) cannot cache a large amount of the mapping table entries, leading to frequent demand loading of the table entries from the slow flash memory.

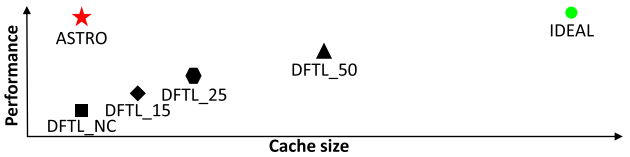


FIGURE 1. Random read performance.

To tackle this critical challenge in the NAND flash-based SSDs, this paper proposes a novel framework called ***ASTRO*** (Address translation framework with Speculative Translation for Read Optimization on SSDs). ASTRO aims to provide an ideal random read performance, which can be achieved when all address mapping information is stored in the mapping table cache, without employing the cache as shown in Figure 1. ASTRO is designed on top of the page-level mapping scheme to achieve high performance and flexibility in address mapping. The key idea behind the ASTRO is to maintain the spatial contiguity of the physical page addresses (PPAs) in the flash memory for the contiguous logical page addresses (LPAs) as much as possible so that most PPAs can be speculatively translated by adding an offset to a base address. ASTRO validates the speculated PPAs by comparing the requested LPA with the embedded LPA stored in the spare region (i.e., OOB, Out-Of-Band) of the page read from the speculated PPAs.

ASTRO consists of three key mechanisms: Lazy Page Ordering (LPO), Speculative Read (SpecREAD), and Contiguity Checking (ContCHECK). LPO divides the physical address space of the flash memory into multiple regions and then rearranges the physical pages in the order of the corresponding logical addresses within each region in the background (e.g., read reclaim or garbage collection time). The second mechanism, SpecREAD, reads the physical pages from the speculated PPA speculatively obtained by adding an offset to a Physical Page Base Address (PPBA) if the page lies in a rearranged region. Then, it validates the speculated PPA to ensure the correctness of the address translation. ContCHECK minimizes mis-speculation in the address translation by checking the updated pages for each rearranged region.

Our experimental results show that ASTRO achieves an average speed up of 1.8x for synthetic random read workloads and 1.34x for real-world read intensive workloads, while minimally impacting the write amplification factor.

II. BACKGROUND AND MOTIVATION

A. ADDRESS TRANSLATION IN SSDS: A KEY CHALLENGE

In Solid-State Drives (SSDs), the Flash Translation Layer (FTL) plays a vital role in managing data stored in the flash memory. It translates logical addresses used by the operating system to physical addresses of the flash memory. This translation is necessary because the flash memory has several unique constraints. First, it is impossible to overwrite existing pages; instead, updated data is written to a page of the erased block, and the old page is marked invalid. Second, the write granularity (i.e., page size) differs from the erase granularity (i.e., block size). Due to this, writing the updated data at the same physical address is inefficient because this approach requires erasing and rewriting the entire data in the block associated with the physical address. Third, the flash cells have limited program/erase (P/E) cycles, necessitating a wear-leveling mechanism that evenly distributes write operations across the flash cells by moving frequently updated data to a less-used block. Recently, the read disturbance problem has become particularly severe as the density of the flash cells has increased [11], [28]. This higher density means that cells are more susceptible to interference from their neighbors during read operations. To mitigate this problem, the FTL moves data from blocks that have been frequently read to an erased block for resetting the read disturbance effect [42].

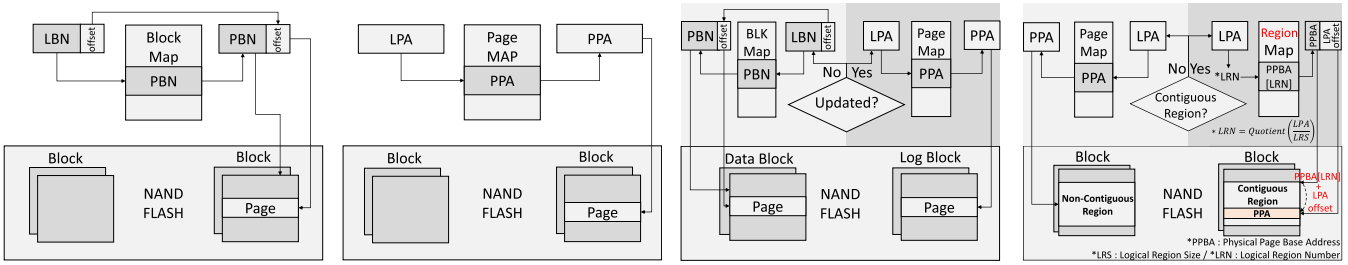
The address translation capability of the FTL offers substantial flexibility in managing the flash memory, as it allows data to be placed and moved dynamically anywhere in the memory. However, this flexibility introduces additional overhead in maintaining the address mapping information, which can result in significant performance degradation.

B. ADDRESS MAPPING SCHEMES FOR SSDS

This subsection describes three conventional address mapping schemes that support the address translation in the FTL and discusses their limitations.

1) BLOCK-LEVEL MAPPING

This scheme maps Logical Block Numbers (LBNs) directly to Physical Block Numbers (PBNs). Because the address mapping information is maintained in block size, this mapping scheme requires a small L2P map. In addition, the



(a) Block-level mapping (b) Page-level mapping (c) Hybrid mapping (d) ASTRO

FIGURE 2. Address mapping schemes.

L2P address translation overhead is low because the physical page is accessed by calculating the LBN and page offset of the logical page as shown in Figure 2a. However, this scheme presents a significant challenge due to an increased Write Amplification Factor (WAF). Since NAND flash cannot be overwritten, each write operation necessitates rewriting an entire block, even if a small portion of data in the block is updated.

2) PAGE-LEVEL MAPPING

To reduce the WAF, a page-level mapping scheme manages data at page granularity by maintaining the mapping between logical page addresses (LPAs) and physical page addresses (PPAs) for each page, as shown in Figure 2b. This mapping scheme allows more efficient use of storage space and longer lifespan of flash cells, as individual pages can be programmed without rewriting the entire block. However, this scheme requires a huge L2P map, which cannot be accommodated in the SSD controller's small on-chip memory. Consequently, the SSDs utilizing the page-level mapping store the L2P map in the flash memory, which can lead to significant performance degradation unless the L2P map is cached in a faster memory such as DRAM or SRAM.

3) HYBRID MAPPING

Hybrid mapping combines the strengths of both block-level and page-level mapping. In Figure 2c, frequently updated data is managed at page granularity to reduce the write amplification while less frequently updated data is managed at block granularity to reduce the L2P map size. In this regard, many hybrid mapping schemes have been proposed [5], [21], [23], [24]. However, these mapping schemes have not overcome the limitations of block level mapping, which reduces performance and shorten the lifespan of NAND when block merge occurs frequently due to random writes.

C. CACHING, IS IT A PANACEA?

1) PAGE-LEVEL MAPPING WITH L2P MAP CACHE

Among the above schemes, commodity SSDs usually employ page-level mapping [10], [13] that has a small impact on the WAF. As described above, if the L2P mapping information is not stored in the embedded DRAM or SRAM, the SSD controller retrieves the mapping information from the flash

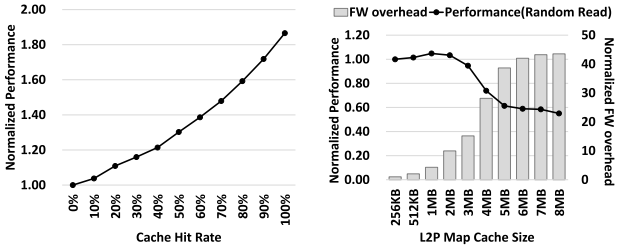


FIGURE 3. Performance impact of L2P map cache hit rate (left) and firmware overhead (right).

memory for every I/O request. To address this critical challenge in the page-level mapping scheme, many enterprise SSDs store the entire mapping table in an L2P map cache typically implemented with DRAMs [12]. Figure 3 (left) shows the random read performance according to the hit rate of the L2P map cache; the experimental methodology is described in section IV-A. As shown in the figure, the performance tends to increase as the cache hit rate increases. The configuration with a perfect cache (i.e., a hit rate of 100%) achieves a speedup of 87% over a cache-less configuration.

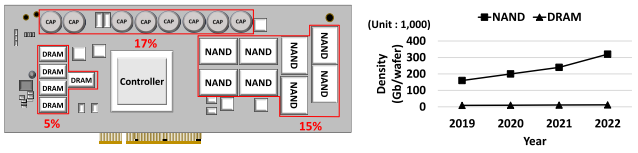


FIGURE 4. Comparison of NAND flash and DRAM: area proportion on an enterprise SSD (left) and density trend (right).

2) LIMITED SCALABILITY

Although caching the L2P map in fast memory effectively reduces address translation overheads, this approach is fundamentally constrained by its limited scalability due to firmware overheads. Increasing the size of the L2P map cache results in longer execution times for cache lookup operations, which are typically implemented in the SSD's firmware (FW). Figure 3 (right) shows the firmware overhead and read performance of the SSD with varying cache sizes. As shown in the figure, a 256KB cache outperforms a 4MB cache.

While using a larger cache generally improves performance by increasing the cache hit rate, the overhead from cache lookup operations also grows with cache size, offsetting the performance benefit of the large capacity.

Hardware cost is another key factor to limit the scalability of the caching approach. Specifically, enterprise SSDs use DRAMs to cache all or a large portion of the L2P map. Incorporating DRAMs in SSDs raises the product price, increases power consumption, and requires substantial space on the circuit board to accommodate the DRAM chips. Figure 4 (left) illustrates a conceptual view of an SSD board composed of NAND flash chips, DRAM chips, an SSD controller, supercapacitors, etc. In an enterprise SSD [8], NAND flash and DRAM occupy 15% and 5% of the board area, respectively. When using DRAMs in the SSDs, supercapacitors also need to be included to provide reliable power until the L2P map in DRAM is stored in flash memory during power loss, consuming another 17% of the board space. Unfortunately, the area overhead of the DRAMs is likely to increase further in the future SSD products because the DRAM is not scaled well compared to the NAND flash memory, as shown in Figure 4 (right). Consequently, many low-cost SSDs, such as DRAM-less SSD [39] and UFS mobile storage [15]), avoid using DRAMs to reduce the hardware costs, fundamentally limiting the adoption of the caching approach.

3) RELIABILITY CONCERN

Recently, the reliability issues with DRAM devices have become increasingly severe [4], [29], [34], [35], which can potentially limit their use as L2P map caches in enterprise SSDs. Errors in DRAM, particularly those affecting the L2P map, can cause significant data management problems within SSDs. For instance, errors in the L2P map can cause data to be written in incorrect locations, leading to data loss or corruption, and can also allow unauthorized data to be read. To address these growing reliability concerns, SSD architectures need to incorporate more advanced error detection and correction mechanisms for DRAM as well as the NAND flash memories [17], [30].

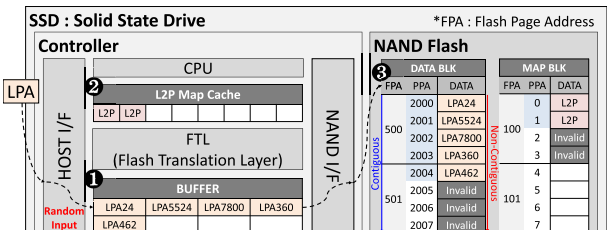


FIGURE 5. Write process of page-level mapping.

D. BUFFERED WRITE SCHEME: MAIN CAUSE OF NON-CONTIGUITY IN ADDRESS MAPPING

Figure 5 illustrates a typical write scheme of the SSDs employing the page-level mapping. The SSD controller stores incoming write data in an internal write buffer in the order it receives from the host (❶). Then, the FTL creates a logical-to-physical mapping in an L2P mapping table cache for each buffered data (❷). Afterward, the data in the write buffer is written to a data block of the NAND flash, and the new L2P mapping is written to a map block (❸). At this time, four pages (i.e., 4KB data) are stored together in a contiguous physical location of the data block. This is because the granularity (e.g., 16KB) of the read and write operations is larger than the logical page size in the modern high-density NAND flash memories [18]; we refer to the smallest unit of read and write operations in NAND flash as a flash page. In the example shown in the Figure 5, four 4KB pages at LPA24, LPA5524, LPA7800, and LPA360 are written to the same 16KB flash page (FPA500 in this example).

We can make two key observations in this buffered write scheme. First, the pages that are not contiguous in the logical address space can be stored contiguously in a physical location of the NAND flash memory. Second, as the write operation of the modern NAND flash is performed in flash page size (e.g., 16KB), which is larger than the logical page size (e.g., 4KB), some 4KB pages within a flash page can be invalid when the flash page is partially written (e.g., PPA2005, 2006, and 2007 are invalid in the example shown in Figure 5).

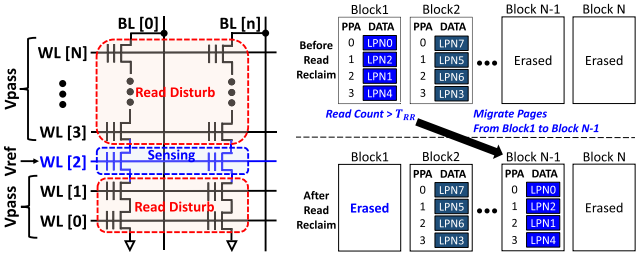


FIGURE 6. Read reclaim scheme (right) to mitigate read disturbance (left) in NAND flash.

E. READ RECLAIM: A HIDDEN OPPORTUNITY

The modern high-density NAND flash memory is vulnerable to the read disturbance problem where read operations on a memory cell cause unintended shifts of the threshold voltages (Vth) of its neighboring cells [3], [11], [33], [41], [42]. As shown in Figure 6 (left), a NAND flash block is organized as an array of NAND memory cells (i.e., floating-gate transistors) that are serially connected. To read a value stored in a memory cell, a reference voltage ($Vref$) is applied to the wordline (i.e., WL2 in the figure) of a selected cell while a read voltage ($Vpass$) is applied to the wordlines of unselected cells. Since the read voltage is relatively high to ensure all unselected cells are turned on, the unselected cell can be unintentionally programmed when the high read voltage is repeatedly applied to the cells, leading to data corruption. In modern high-density NAND, the read voltage affects more unselected cells. The read disturbance-induced errors are a crucial reliability concern especially for read-intensive workloads as the error rate depends on the frequency of the read operations [33].

To mitigate the read disturbance, the SSDs with high-density NAND flash memories employ a Read Reclaim (RR) scheme (a.k.a read refresh) that migrates pages in a specific block to another block when the read count of the block exceeds a threshold value (T_{RR}) [11], [26], [33], [42]. As all pages are reprogrammed in a new block, any disturbance effects on those pages are eliminated. Figure 6 (right) illustrates an example of the RR process. In this example, since the read count of block1 is larger than T_{RR}, all its contents are copied to an erased block (Block N-1), and then block1 is erased. The RR process can significantly impact the performance of SSDs and reduce the lifetime of the NAND flash as it involves additional write operations.

While the RR process introduces additional overhead to SSDs, it also offers an opportunity to improve address translation. As described in Section II-D, the current write scheme for the SSDs with page-level mapping results in non-contiguity in the address mappings for pages. However, during the RR process, if pages are rearranged as they're migrated to a new block, such that contiguous LPAs are mapped to contiguous PPAs, address translation can be performed without using the L2P map. This novel optimization will be elaborated on in the subsequent section.

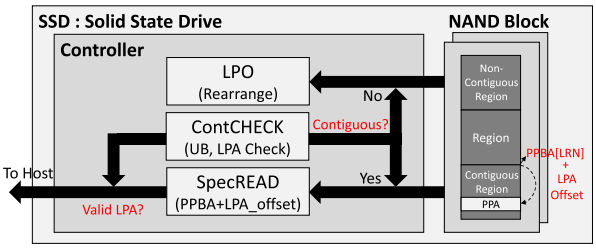


FIGURE 7. ASTRO framework.

III. ASTRO FRAMEWORK

A. AN OVERVIEW

In this paper, we propose ASTRO, a novel framework designed to minimize the address translation overhead using speculation rather than traditional caching methods. Figure 7 shows an overall architecture of ASTRO, which is comprised of three mechanisms: Lazy Page Ordering (LPO), Speculative Read (SpecREAD), and Contiguity Checking (ContCHECK).

LPO partitions the physical address space into regions and rearranges the flash pages so that the pages within each region are ordered based on their LPA. SpecREAD speculatively translates LPAs to PPAs by assuming the pages are ordered in an LPA sequence and then reads data from the speculated PPAs. Finally, ContCHECK monitors any updates on the rearranged region to minimize mis-speculation in the speculative address translation. ASTRO is implemented as part of FTL firmware, and some features are optimized through simple hardware implementations within the SSD controller.

B. LAZY PAGE ORDERING

As described above, in a page-level mapping, new pages are stored in flash memory in the order they arrive. As a result, they are assigned arbitrary PPAs, which leads to no predefined sequence or structure governing where the pages get placed in the flash memory. While this approach efficiently utilizes the storage space, it does not necessarily optimize spatial locality for I/O requests on SSDs.

LPO plays a crucial role in ASTRO by rearranging the pages based on their LPAs to optimize the placement of pages within the flash memory. By rearranging the pages into a structured arrangement, LPO creates a more predictable environment for the address translation, thereby allowing the SpecREAD to perform its speculative translation process more efficiently and accurately. LPO is basically executed in conjunction with the Read Reclaim (RR) to minimize additional data copies. Since the RR process is an essential operation in the NAND flash and it involves extensive data copies, integrating LPO into the RR process can reduce data copies and mitigate performance impacts associated with LPO.

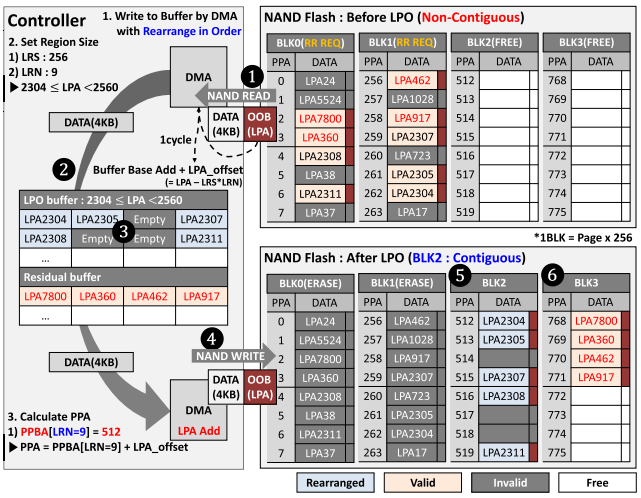


FIGURE 8. Lazy page ordering.

Figure 8 illustrates the LPO's rearranging process with an example scenario. LPO arranges the valid pages in the ascending order of LPA and keeps the space of the invalid LPAs empty. To this end, LPO partitions the logical address space into logical regions and count the number of valid pages per the logical region. Then, it selects an unordered region with the largest number of valid pages for the page rearrangement. In the example shown in Figure 8, it is assumed that the logical region size (LRS) is 256 pages and LPO selects the logical region with a logical region number (LRN) of 9.

LPO retrieves valid pages belonging to the selected logical region when the pages are copied during a read reclaim process, and then it stores the pages in ascending order of LPA in an LPO buffer (❶). In this process, the buffer addresses for the pages are calculated by adding the pages' LPA offset to the base address of a rearranging area in the LPO buffer (❷).

The LPA offset is obtained from the LPA information stored in the OOB area of the NAND flash. This address calculation method makes the memory spaces for invalid LPAs remain empty in the LPO buffer. For example, the page at LPA2306 is invalid, and thus, the corresponding location for the page in the LPO buffer remains empty (❸). After the pages are rearranged within the LPO buffer, they are written back to free blocks (❹). In this example, the rearranged pages are written to the block 2 which was a free block (❺).

The region information is maintained with a Region Map Table (RMT) as shown in Figure 9. Each entry in the table is comprised of the Logical Region Number (LRN), Physical Page Base Address (PPBA), Contiguous Region Bit (CRB), and Update Bitmap (UB). PPBA is the physical address where the corresponding region's pages are stored in the flash memory. CRB indicates whether the corresponding region have been rearranged through the LPO process. UB is used to determine whether specific pages are updated.

While valid pages in a block are migrated during an RR process, the block may contain pages that do not belong to the selected region for the rearrangement. LPO temporarily stores those pages in a residual buffer and then writes them to a free block when the residual buffer is almost full (❻). These residual pages are rearranged later when they are migrated in the future RR process.

As described above, LPO requires two buffers: LPO and residual buffers. The LPO buffer size varies depending on the logical region size (LRS). In the example shown in Figure 8, since the LRS is set to 256 pages, the LPO buffer size should be at least 1MB (256 × 4KB). The residual buffer size can be a multiple of block size, which can be configured by the SSD firmware.

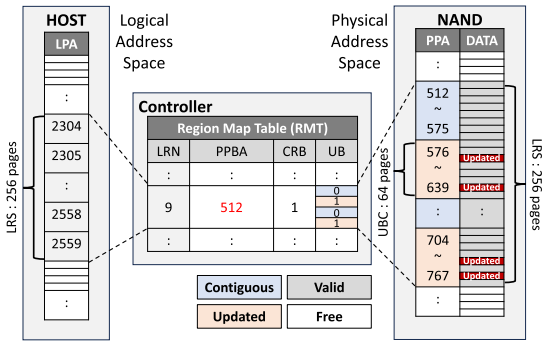


FIGURE 9. Region map table.

C. SPECULATIVE READ

By using LPO to rearrange the pages, logically adjacent pages become physically adjacent within the flash memory. This structured arrangement of pages allows for the straightforward calculation of PPAs for most pages, typically requiring only a simple addition of the base address and offset (equation 1). If ASTRO can correctly speculate the PPAs for most pages, it will significantly reduce the flash memory accesses for loading the L2P mapping table, resulting in a significant improvement in read performance.

Speculative Read mechanism (SpecREAD) computes the LRN for the given logical address using the formula outlined in equation 2. For example, in Figure 9, if the Logical Region Size (LRS) is 256, the pages with LPA between 2304 and 2559 fall under LRN 9. SpecREAD then obtains the Physical Page Base Address (PPBA) corresponding to this LRN from the RMT. Subsequently, SpecREAD determines a PPA by adding the PPBA and an LPA offset ($Offset_{LPA}$) calculated using equation 3. As shown in Figure 8, it is assumed that the PPBA for the LRN 9 is 512. Thus, SpecREAD translates the LPA 2307 into the PPA 515 by adding the LPA offset of 3 to the PPBA of 512.

$$PPA = PPBA[LRN] + Offset_{LPA} \tag{1}$$

$$LRN = Quotient(LPA/LRS) \tag{2}$$

$$Offset_{LPA} = LPA - LRN \times LRS \tag{3}$$

SpecREAD calculates the PPAs by assuming that the logically adjacent pages are also contiguously stored in the flash memory. However, this assumption can fail when data in the rearranged region is updated. When data needs to be updated, the new data is written to a free page rather than overwriting the existing one. The old page is then marked as invalid, and the new page takes its place in the L2P mapping table as described in section II-A. This "out-of-place" update breaks the ordered sequence of the rearranged regions. Therefore, it is necessary to validate the calculated PPAs to ensure the correct read operations.

To this end, ASTRO stores the LPA for a page in its OOB area. The OOB area mainly stores parity for ECC, but LPA can also be stored. Figure 10 shows the overall operation flow of ASTRO. Once the SpecREAD reads a page from the speculated PPA, it compares the LPA fetched from the corresponding OOB with the given LPA from the host system. If the two LPAs do not match, SpecREAD obtains the actual PPA for the given LPA by loading the corresponding L2P mapping table entry from the flash memory. It then reaccesses the flash memory to read the page stored at this correct PPA.

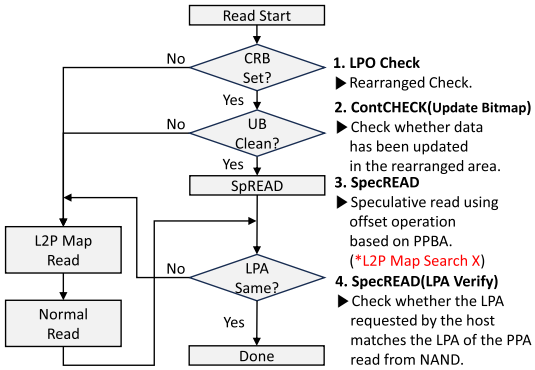


FIGURE 10. The flow chart detailing the operation of ASTRO.

D. CONTIGUITY CHECKING

In the ASTRO framework, minimizing the mis-speculation in the address translation is essential for performance. To achieve this goal, the ContCHECK mechanism monitors

the updates in the logical regions. While it is possible to monitor these updates using a dirty bitmap, where each bit corresponds to a 4KB page in the flash memory, this method comes with a significant memory overhead. For instance, monitoring a 512GB SSD would require 16MB of memory. To be more efficient, ContCHECK uses an Update Bitmap (UB) maintained in the RMT for each logical region. The UB is a sort of bloom filter that is a probabilistic data structure that can quickly determine whether an element belongs to a set. Using the UB, ContCHECK efficiently determines the presence of a requested page within the rearranged region with a small hardware overhead. In Figure 9, since some pages between PPA 576 to 639 and between PPA 704 to 767 are updated, the UB of LRN9 is set to 4'b0101. The size of a UB varies depending on how many pages are allocated per a 1-bit of the UB. For example, if the total capacity of the SSD is 512GB, there are 128M pages. Therefore, if a 1-bit of the UB covers 64 pages, the total size of UBs in the RMT is 256 KB.

TABLE 1. Summary of SSD configuration.

Parameter		Value
NAND Parameter	tR	40us
	tPROG	200us
	tERASE	2ms
LPO Parameter	Read Reclaim Threshold (T_{RR})	1024
	Read Threshold of LRS ($T_{LPOread}$)	T_{RR} of LRS
	Update Threshold of LRS ($T_{LPOupdate}$)	25% of LRS
Number of pages per block		1024 (4MB)
Logical Region Size (LRS)		4096 (16MB)
Update Bitmap (UB) coverage		64 / 256 / 1024 / 4096
Total SSD capacity		64GB

IV. EVALUATION

A. METHODOLOGY

1) SSD SIMULATOR

We use an SSD simulator called FEMU [25], a QEMU-based flash emulator designed to facilitate full-stack software/hardware SSD research. The experiment employs a Black Box SSD model with modified FTL code and NAND flash parameters. We assume the read latency of NAND flash is 40us [16]. In the case of demand loading, a data page is read after loading the mapping table entries from the flash memory. On the other hand, SpecREAD directly reads a data page if it is expected to lie in the rearranged region. We evaluate the impact on performance using the UB coverage of 64, 256, 1024, and 4096 pages, with a default of 64 pages. We assume the LRS is 4096 pages ($4096 \times 4KB = 16MB$). The simulated SSD performs read reclaim (RR) when 1024 pages are read from a block [27]. We configure ASTRO to perform the LPO in two situations: when a read reclaim occurs and when the update count in the logical region exceeds 25% of the LRS. The configurations of the simulated SSD are summarized in Table 1.

TABLE 2. Summary of workloads.

Case	Random Read					
Benchmark	FIO			Filebench		
I/O size	4KB			4KB		
Test size	5GB			5GB		
Thread count	5			5		
R/W ratio	100:0			100:0		
Run time	300s			300s		
Case	Filebench					
Type	Fileserver	Webserver	Webproxy	Varmail		
Operation	Heavy data transfer	Logging client-access records	Intermediary between client and server	e-mail check		
I/O size(r/w)	16KB/16KB	1MB/8KB	1MB/16KB	1MB/16KB		
Test size	5GB	5GB	5GB	5GB		
Thread count	5	5	5	5		
R/W ratio	1:2	10:1	5:1	1:1		
# of operation	100M	100M	100M	100M		
Case	YCSB					
Type	A	B	C	D	E	F
Operation	Update heavy records	Read mostly records	Read only records	Read latest records	Short ranges of records	Read-modify-write
Update	50%	5%	0%	0%	0%	0%
Insert	0%	0%	0%	5%	5%	0%
Read	50%	95%	100%	95%	0%	50%
Scan	0%	0%	0%	0%	95%	0%
RMW	0%	0%	0%	0%	0%	50%
I/O size	4KB	4KB	4KB	4KB	4KB	4KB
Test size	5GB	5GB	5GB	5GB	5GB	5GB
Thread count	5	5	5	5	5	5
R/W ratio	50:50	95:5	100:0	95:5	95:5	50:50
# of operation	100M	100M	100M	100M	100M	100M

2) WORKLOADS

We use synthetic random read workloads and real-world workloads. The random read workloads are our main target, as the performance of these applications have many read request without write and we can exploit this characteristic to significantly improve the performance of read by rearranging physical pages in the order of the corresponding logical addresses within each region in read reclaim. In this workload, we test a 5GB workload with FIO [2] and Filebench(FB) [36] benchmark suit. This workload writes 5GB of data to the SSD and then reads 4KB data at random addresses. Table 2 (top) summarizes the I/O characteristics of these workloads.

Regarding the real-world workloads, we also test ten workloads via the Filebench(FB) and YCSB [6] benchmark suits, whose detailed parameters are listed in Table 2. The goal of this evaluation is to demonstrate that the impact of performance brought by our proposal is quite big, even under read-intensive workloads with a few write operation. Note that the number of operation across all workloads is 100M. In Filebench, Table 2 (middle) shows four types of this real-world workload: Fileserver, Webserber, Webproxy, Varmail. The Fileserver workloads include file creation, opening, reading, adding, and deleting, requiring heavy updates and a read-to-write ratio of 1:2. The Webserver workload responds to HTTP requests by opening HTML files and periodically updating client access records to show read-to-write characteristics with a 10:1. The Webproxy, operating as

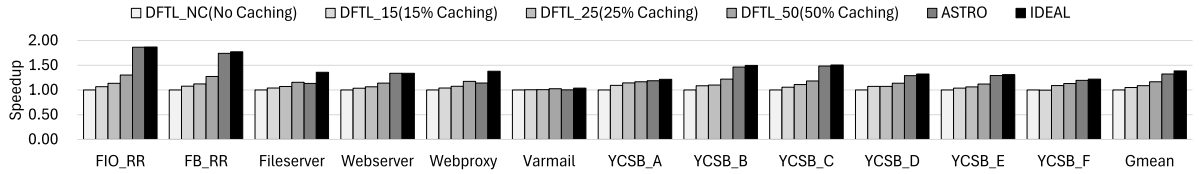


FIGURE 11. Performance of ASTRO compared to the DFTL according to L2P map caching ratio.

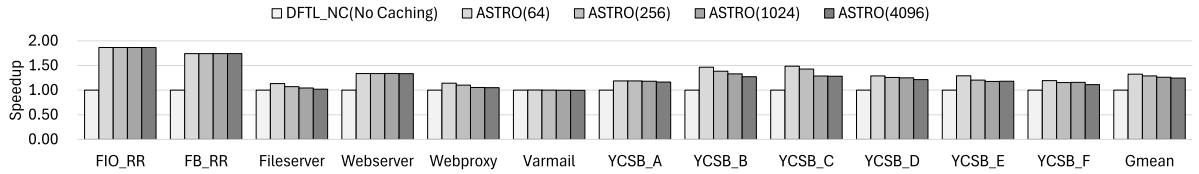


FIGURE 12. Performance impact of update bitmap Coverage.

an intermediary between client and server, executes file read, create, and delete operations, predominantly characterized by reads, with a read-to-write ratio of 5:1. Varmail's e-mail check operations include reading, opening, marking as read, and synchronization. This workload has a 1:1 ratio of read and write [20], [37].

In YCSB, there are six types of real-world workload: A, B, C, D, E, and F [40]. Each type comprises operations such as update, insert, read, scan, and read-modify-write (RMW). "Insert" means write a new record and "Update" means a record by replacing the value of one field. "Read" means reading a record, either one randomly chosen field or all fields, and "Scan" records in order, starting at a randomly chosen record key. The number of records to scan is randomly chosen. "RMW" is an operation that reads, modifies, and updates the record as it is. The detailed description of the YCSB is summarized in Table 2 (bottom). The DB uses RocksDB [7], [9], an unstructured database (NOSQL) developed by Facebook based on LevelDB, Google's open source project. Because it is NOSQL (Not Only SQL), it uses a key-value storage method, making it suitable for processing large amounts of data. In particular, it is optimized for high performance on SSD storage devices because it has an LSM (Log Structured Merge)-Tree structure.

B. PERFORMANCE

1) RANDOM READ WORKLOADS

In Figure 11, FIO_RR and FB_RR present the performance on random read workloads without the write operation across three novel mechanisms of ASTRO. As can be seen from these results, ASTRO can achieve a speedup level which is as good as the IDEAL. This because, the LPO successfully rearranges the physical pages in the order of the corresponding logical addresses within each region in read reclaim, and as a result, ASTRO utilizes the rearranging to improve the performance and minimize the ratio of demand load, as shown in Figure 13b. ASTRO achieves speedup and SpecREAD ratio of 1.87x, 99% for FIO_RR (4KB) and 1.74x, 99% for FB_RR (4KB) compared to the baseline (DFTL_NC [10]).

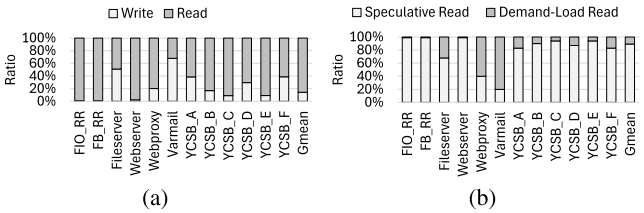


FIGURE 13. (a) Breakdown of I/O requests and (b) Proportion of speculative read.

2) REAL-WORLD WORKLOADS

To demonstrate the feasibility of our proposal in a real-world workload, we test it on Filebench (FB) and YCSB. The real-world workloads are broken into three categories: (1) read-intensive, (2) write-intensive, and (3) non-intensive. The read-intensive workloads are Webserver, Webproxy, YCSB_B,C,D and E with a few write operation and the ratio of reads is 98%, 80%, 83%, 91%, 70% and 91% in Figure 13a. Figure 11 presents that ASTRO can achieve close to IDEAL speedup, higher than DFTL_50(50% Caching), except for Webproxy. This is because the LPO successfully rearrange the physical pages in ascending order of LPA by read reclaim even when there is an update due to a few of write. Hence, the $T_{LPOupdate}$ has to be set to the appropriate value (i.e. 25% of LRS) to ensure that the speedup is not significant degraded.

Interestingly, despite the 80% read operation ratio of the Webproxy, the reason for the low speedup is that updates occur repeatedly on the specific logical address region [20]. Due to such repeated writes to the specific logical address region, the speedup of ASTRO with SpecREAD is slightly lower than the speedup observed with DFTL_50. In Figure 11 and 13b, the speedup and SpecREAD ratio of read-intensive workload are 1.34x, 99% for Websrever, 1.14x, 40% for Webproxy, 1.47x, 90% for YCSB_B, 1.49x,

94% for YCSB_C, 1.29x, 87% for YCSB_D, 1.29x, 94% for YCSB_E.

The write-intensive workloads are Fileserver and Varmail, with write ratios of 51% and 68%, respectively. In Figure 11, for write-intensive workloads, the performance gains are higher or similar to DFTL_25, but less than DFTL_50. This is because the LPO cannot successfully rearrange the physical pages in ascending order of the LPA due to the high number of writes and low read reclaim. Also, even if the rearranging is completed, the contiguity is easily broken due to frequent updates and the UB of ContCHECK is set to 1, which reduces the SpecREAD ratio, as shown in Figure 13b. In this case, the performance can be improved by setting the $T_{LPOread}$ value smaller (i.e. 50% of T_{RR}) so that the rearrange occurs more frequently, but the WAF due to the LPO can increase. The speedup and SpecREAD ratio of write-intensive workload are 1.13x, 68% for Filesrever and 1.0x, 20% for Varmail, as shown in Figure 11 and 13b.

The non-intensive workloads are YCSB_A and F, with write ratios of 39% both. In Figure 11, for non-intensive workloads, the speedup is slightly higher than DFTL_50. This is because the rearrange contiguity and SpecREAD ratio by LPO are higher than write-intensive workload and lower than read-intensive workload because the write and read ratios are similar. As a result, the speedup and SpecREAD ratio of non-intensive workload are 1.19x, 83% for both YCSB_A and YCSB_F, as shown in Figure 11 and 13b.

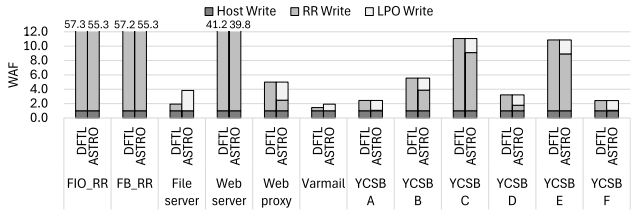


FIGURE 14. Write amplification factor.

C. WRITE AMPLIFICATION FACTOR

Figure 14 shows the WAF of DFTL and ASTRO for all workloads. In random read and read-intensive workload, the WAF by RR of DFTL is basically high. This is because there are many read requests, so a lot of read reclaim occurred. Since ASTRO performs LPO by $T_{LPOread}$ when read reclaim occurred, there is no additional WAF compared to DFTL. On the other hand, in the write-intensive workload, the WAF by the RR of DFTL is basically low. This is because there are many write requests, so less read reclaim occurred. Therefore, in ASTRO, the WAF by LPO occurs not only by $T_{LPOread}$ but also by $T_{LPOupdate}$ additionally. Finally, in non-intensive workload, the WAF between the RR of DFTL and the LPO of ASTRO is almost similar. This is because the write and read ratio is similar, so LPO by $T_{LPOupdate}$ and $T_{LPOread}$ is also performed at a similar ratio. For all workloads, the WAF by Host is 1, and the WAF by LPO is 1.9 for FIO_RR, 1.9 for FB_RR, 2.8 for Fileserver, 1.4 for Webserver, 2.5 for Webproxy, 0.9 for Varmail, 1.4 for YCSB_A, 1.7 for YCSB_B, 2.0 for YCSB_C, 1.4 for YCSB_D, 1.9 for YCSB_E and 1.4 for YCSB_F.

TABLE 3. Hardware overhead.

28n Process(@500MHz)		Baseline	ASTRO	Overhead
Power	Cell Internal Power	77%	77%	0.00%
	Net Switching Power	23%	23%	
	Total Dynamic Power(uW)	358	371	3.63%
Area	Total Cell Area	4345	4704	8.26%

D. PERFORMANCE IMPACT OF UB COVERAGE

Figure 12 shows the performance impact of update bitmap coverage (UBC). The UBC indicates how many pages are allocated to 1 bit of update bitmap (UB). Therefore, if there is almost no write, the speedup is the same regardless of UBC. This is because there is almost no writing, so the probability of contiguity being broken by update is very low regardless of UBC. Therefore, FIO_RR, FB_RR, and Webserver show the same speedup regardless of UBC.

On the other hand, if there is a certain amount of write, the larger the UBC, the lower the speedup. This is because there is a write, so the larger the UBC, the greater the probability that the contiguity will be broken by update. Once the continuity is broken in a specific logical region, the UB of the ContCHECK is set to 1. Then, until it is rearranged through the LPO, the speedup may be reduced by performing demand load read rather than SpecREAD. Therefore, compared to UBC 64, the UBC 4096 has a speedup decrease of 10% for Fileserver, 8% for Webproxy, 0% for Varmail, 2% for YCSB_A, 13% for YCSB_B, 14% for YCSB_C, 6% for YCSB_D, 8% for YCSB_E, 7% for YCSB_F and 6% for Gmean.

E. IMPACT OF HARDWARE-ACCELERATED LPO

When rearranging the pages with FTL firmware in the ARM processor, which is used in many SSD products, 37 cycles are added to the latency of data movement with the DMA engine due to the operation of checking the LPA and setting the address. This corresponds to approximately 74ns based on a Core Frequency of 500 MHz. Assuming that 8MB pages (4KB x 2048) are rearranged, it can be seen that 151.522us (74ns x 2048) is added to the data movement time. This overhead can be eliminated by extending the DMA engine to calculate the address in 1cycle automatically.

F. HARDWARE COST

Table 3 shows the hardware overhead of the extended DMA engine. We synthesize the RTL design using the Synopsys design compiler and UMC 28nm cell library at 500MHz. The area of the DMA hardware increases by about 8.26%. The major contributor to the hardware overhead is the additional register to hold the DMA state to check the LPA in the OOB of the physical page read from NAND. When setting the DMA address, the extended DMA engine checks the LPA offset and performs an operation to add the corresponding offset to the based address of the rearranging area in the buffer.

V. RELATED WORK

A. CACHING L2P MAPPING TABLE

Page-level mapping requires connecting all LPAs to PPAs, so the L2P map size is large. For fast performance, all L2P map must be cached. DFTL [10] selectively caches page-level address mappings. HPB [15] is an integrated host-SSD mapping table management method to increase the effective capacity of the L2P map cache and utilizes host DRAM as the L2P map cache. In [22], a technique is proposed to manage the host memory more efficiently when it stores the L2P mapping table alongside the normal host data. Even if these approaches efficiently enlarge the L2P map cache size, they require modification in the host software and may degrade the application performance due to the reduced memory space that can be allocated to the host applications. ASTRO does not require any modification in the host software and additional L2P map cache resources. In addition, HPB improves random read performance by 1.57x to 1.67x by using Host DRAM, while ASTRO improves random read performance by an average of 1.8x with simple operations without Host DRAM.

B. L2P MAP MANAGEMENT

In constructing the L2P mapping table, SHRD [19] proposed a new address reconstruction technique that converts random write requests into sequential write requests. This approach improves the spatial locality of IO requests to the storage. A Hash-based space-efficient page-level FTL [32] was proposed as a space-efficient method that uses a hash function in address translation. Probability-based address translation for Flash SSDs [14] was proposed as a new probability-based address translation algorithm. In those prior works, unlike existing conversion techniques that maintain accurate L2P mapping information, a probabilistic data structure such as a bloom filter is used. With this approach, the amount of DRAM used for address conversion can be reduced by 20%. However, these approaches still require frequent search operations for L2P mapping information. ASTRO has no overhead for address translation as long as the contiguity of pages is maintained.

VI. CONCLUSION

The L2P mapping table is essential for the NAND flash-based SSDs. However, handling the table can be a critical performance bottleneck as it is typically stored in slow flash memory. A simple approach to mitigate this problem is to cache the table in embedded DRAM. Unfortunately, however, as SSD capacity gradually increases, the L2P mapping table size also increases significantly. As a result, resource and power risks are increasing as the size of embedded DRAM required for cache increases.

To tackle this fundamental limitation of the NAND flash-based SSDs, this paper proposes ASTRO, a framework to translate LPAs to PPAs without accessing the L2P mapping table as much as possible. ASTRO rearranges pages in increasing order of the LPA when performing read reclaim. Therefore, the increase in additional WAF for LPO can be minimized. By doing this, ASTRO can translate the LPAs to PPAs with a simple addition operation. Our experimental results demonstrate that ASTRO achieves an average 1.8x performance improvement comparable to the configuration with an ideal caching mechanism in random read workload. And ASTRO improves performance by an average of 1.34x on real-world read-intensive workloads and an average of 1.32x across all workloads compared to the IDEAL Gmean value of 1.38x.

REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2008, pp. 57–70.

[2] J. Axboe. (2005). *Fio-flexible I/o Tester Synthetic Benchmark*. Accessed: Jun. 13, 2015. [Online]. Available: https://github.com/axboe/fio

[3] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.

[4] J. Chen, X. Jiang, Y. Zhang, L. Liu, H. Xu, and Q. Liu, "CARE: Coordinated augmentation for elastic resilience on DRAM errors in data centers," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 533–544.

[5] H. Cho, D. Shin, and Y. Ik Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 507–512.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, Jun. 2010, pp. 143–154.

[7] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum, "Optimizing space amplification in RocksDB," in *Proc. CIDR*, Jan. 2017, pp. 1–6.

[8] (2024). *Enterprise SSD*. [Online]. Available: https://semiconductor.samsung.com/kr/ssd/enterprise-ssd/pm1733-pm1735/

[9] *Rocksdb, A Persistent Key-value Store for Fast Storage Enviroments*, Facebook, Menlo Park, CA, USA, 2019.

[10] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, Feb. 2009.

[11] K. Ha, J. Jeong, and J. Kim, "An integrated approach for managing read disturbs in high-density NAND flash memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 7, pp. 1079–1091, Jul. 2016.

[12] K. Han, H. Kim, and D. Shin, "WAL-SSD: Address remapping-based write-ahead-logging solid-state disks," *IEEE Trans. Comput.*, vol. 69, no. 2, pp. 260–273, Feb. 2020.

[13] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang, "Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–12.

[14] J. Im, H. Kim, Y. Won, J. Oh, M. Kim, and S. Lee, "Probability-based address translationfor flash SSDs," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 97–100, Jul. 2020.

[15] W. Jeong, H. Cho, Y. Lee, J. Lee, S. Yoon, J. Y. Hwang, and D. Lee, "Improving flash storage performance by caching address mapping table in host memory," in *Proc. 9th USENIX Workshop Hot Topics Storage File Syst.*, Jan. 2017, pp. 1–6.

[16] K. Kawai et al., "A 1Tb density 3b/Cell 3D-NAND flash on a 2YY-tier technology with a 300MB/s write throughput," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2024, pp. 244–246.

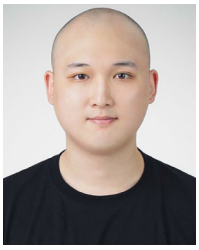
[17] B. S. Kim, J. Choi, and S. L. Min, "Design tradeoffs for SSD reliability," in *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*, Jan. 2019, pp. 281–294.

[18] C. Kim et al., "A 512-gb 3-b/Cell 64-stacked WL 3-D-NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 124–133, Jan. 2018.

[19] H. Kim, D. Shin, Y. H. Jeong, and K. H. Kim, "SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, 2017, pp. 271–284.

[20] H. Kim and S. Hong, "Why address translation matter? Analyzing page access patterns in NAND flash-based SSDs," in *Proc. Int. Tech. Conf. Circuits/Syst., Comput., Commun. (ITC-CSCC)*, Jun. 2023, pp. 1–5.
[21] J. Kim, J. Min Kim, S. H. Noh, S. Lyul Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.
[22] Y. Kim, I. Choi, J. Park, J. Lee, S. Lee, and J. Kim, "Integrated host-SSDmapping table management for improving user experience of smartphones," in *Proc. 21st USENIX Conf. File Storage Technol. (FAST)*, 2023, pp. 441–456.
[23] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, p. 18, Jul. 2007.
[24] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 6, pp. 36–42, Oct. 2008.
[25] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The case of FEMU: Cheap, accurate, scalable and extensible flash emulator," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2018, pp. 83–90.
[26] J. Li, B. Huang, Z. Sha, Z. Cai, J. Liao, B. Gerofi, and Y. Ishikawa, "Mitigating negative impacts of read disturb in SSDs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 1, pp. 1–24, Sep. 2020, doi: 10.1145/3410332.
[27] C.-Y. Liu, Y.-M. Chang, and Y.-H. Chang, "Read leveling for flash storage systems," in *Proc. 8th ACM Int. Syst. Storage Conf.*, May 2015, pp. 1–10.
[28] C.-Y. Liu, Y. Lee, M. Jung, M. T. Kandemir, and W. Choi, "Prolonging 3D NAND SSD lifetime via read latency relaxation," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2021, pp. 730–742.
[29] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 415–426.
[30] N. R. Mielke, R. E. Frickey, I. Kalastirsky, M. Quan, D. Ustinov, and V. J. Vasudevan, "Reliability of solid-state drives based on NAND flash memory," *Proc. IEEE*, vol. 105, no. 9, pp. 1725–1750, Sep. 2017.
[31] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD failures in datacenters: What? When? And Why?" in *Proc. 9th ACM Int. Syst. Storage Conf.*, Jun. 2016, pp. 1–11.
[32] F. Ni, C. Liu, Y. Wang, C. Xu, X. Zhang, and S. Jiang, "A hash-based space-efficient page-level FTL for large-capacity SSDs," in *Proc. Int. Conf. Netw., Archit., Storage (NAS)*, Aug. 2017, pp. 1–6.
[33] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-cosmos: In-flash bulk bitwise operations using inherent computation capability of NAND flash memory," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 937–955.
[34] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 297–310, May 2015.
[35] V. Sridharan and D. Liberty, "A study of DRAM failures in the field," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2012, pp. 1–11.
[36] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *USENI*, vol. 41, no. 1, pp. 6–12, 2016.
[37] J. Xu and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2016, pp. 323–338.
[38] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of NVMe SSDs and their implication on real world databases," in *Proc. 8th ACM Int. Syst. Storage Conf.*, May 2015, pp. 1–11.
[39] S. Yang, "Improving the design of dram-less pcie SSD," in *Proc. Flash Memory Summit*, 2017, pp. 1–16.
[40] YCSB. (2020). *YCSB Core Workloads*. [Online]. Available: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads
[41] K. J. Yoon, Y. Kim, and C. S. Hwang, "What will come after V-NAND—Vertical resistive switching memory?" *Adv. Electron. Mater.*, vol. 5, no. 9, Sep. 2019, Art. no. 1800914.
[42] G. Zhang, Y. Deng, Y. Zhou, S. Pang, J. Yue, and Y. Zhu, "Cocktail: Mixing data with different characteristics to reduce read reclaims for nand flash memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2336–2349, Jul. 2023.

HYUNGJIN KIM is currently pursuing the master's degree with the Department of Semiconductor and Display Engineering, Sungkyunkwan University, South Korea. In 2013, he joined Samsung Electronics Corporation, Hwasung, South Korea, where he was involved in the design of a controller. His current research interests include storage systems and flash translation layer (FTL).



SEONGWOOK KIM (Graduate Student Member, IEEE) received the bachelor's and master's degrees. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. His research interests include deep learning accelerator architecture and memory compression.

JUNHYEOK PARK (Graduate Student Member, IEEE) is currently pursuing the master's degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. His current research interests include memory systems for heterogeneous computing platforms, virtual memory, and GPU microarchitecture and systems.

GWANGEUN BYEON (Graduate Student Member, IEEE) is currently pursuing the integrated master's and Ph.D. degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. His research interests include DNN accelerators and sparse matrix computations.

SEOKIN HONG (Member, IEEE) received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2015. From 2015 to 2017, he was a Senior Engineer with Samsung Electronics. During his two years there, he was involved in a project that developed the 3D stacked memory. In 2017, he moved to the IBM T. J. Watson Research Center, where he worked on secure processor architectures and emerging memory/storage systems. He is currently an Associate Professor with Sungkyunkwan University, South Korea. His current research interests include the design of low-power, reliable, and high-performance processor architectures and memory systems. He received best paper awards from the International Conference on Computer Design (ICCD), in 2010, and the Design Automation and Test in Europe (DATE), in 2013.