

FSPDA: A Full Sequence Program Data Allocation Scheme for Boosting 3-D NAND Flash Read Performance

Shujie Pang^{ID}, Yuhui Deng^{ID}, Zhaorui Wu^{ID}, Genxiong Zhang^{ID}, Jie Li, and Xiao Qin^{ID}

Abstract—Multibit 3-D NAND flash-based solid-state disks (SSDs), offering high storage density, contain multiple types of pages to accommodate multiple bits per physical cell. Full sequence program or FSP can program multiple pages in a word line at a time, thereby improving write throughput. Unfortunately, large-grained FSP operations coarsely aggregate consecutive logical pages on the same word line, which adversely affects the parallelism and latency of read requests. Moreover, FSP smooths the program latencies for different types of pages, whereas the pages still exhibit various read latencies. Multiple read latencies and lower read parallelism noticeably deteriorate the completion efficiency of read requests: SSD performance is degraded. To address this issue, we propose an FSP data allocation scheme called *FSPDA* that incorporates the physical structure characteristics of multibit 3-D NAND, aiming to bolster the read performance of 3-D NAND Flash-based SSDs. *FSPDA* embraces two distinctive and vital features. First, according to the distance between logical pages, *FSPDA* allocates logical pages to specified parallel units and stipulates that consecutive logical pages must be assigned to different planes, thus improving read parallelism and data locality. Second, to further reduce read latency, *FSPDA* employs cache hits to determine hot and cold data to be placed to low-latency and high-latency pages, respectively. We compare *FSPDA* with two state-of-the-art schemes—*OSPADA* and *single-operation-multiple-location*—in terms of multiplane read (MPR) counts, read response time, and GC counts under eight real-world workloads. The experimental results show that compared with the existing schemes, *FSPDA* slashes the number of MPR counts, read response time, and the number of GC counts by an average of 34.4%, 28.5%, and 13.6%, respectively.

Index Terms—3-D NAND flash, data allocation, full sequence program (FSP), multiplane operation.

I. INTRODUCTION

NAND flash-based solid-state disks (SSDs) are favored by consumers and manufacturers thanks to prominent read/write performance. Since the physical structure of 2-D NAND flash limits SSD capacity, 3-D NAND flash becomes a mainstream storage device in the market because of its large capacity and high scalability [1], [2], [3], [4]. The 3-D NAND technology significantly increases the density of NAND-based SSDs by stacking multiple layers of planar NAND. Recently, some 3-D NAND flash chips were increased to 128 layers. Additionally, to further expand the capacity of flash memory, SSD manufacturers have developed multibit 3-D NAND flash, which stores extra bits of data in a NAND cell. For example, triple-level cell (TLC) store three bits information per cell. TLC has become a key storage medium for modern SSDs by virtue of its low price and high storage capacity.

High-density multibit flash, however, significantly increases the latency of programming/reading one-bit data, and multiple bits in a cell usually have various latencies. In particular, a TLC cell has three bits, least significant bit (LSB), central significant bit (CSB), and most significant bit (MSB), exhibiting obviously different program/read latencies. 3-D TLC NAND flash designs these bits into three types of pages with diverse program/read latencies, namely, LSB pages, CSB pages, and MSB pages. In this case, because NAND flash-based SSDs specify the size of a physical page as the unit of program and read operations, the latency gap between the three pages impacts on the overall performance of multibit 3-D NAND flash-based SSDs. To address this latency issue, a growing number of SSD manufacturers implement a full sequence program (FSP) operation to smooth various program latencies. Unlike the traditional program operation, FSP programs all bits in a cell *simultaneously*. Specifically, TLC flash employs the FSP operation to program three types of pages on a word line at the same time. The FSP operation performed on a TLC cell gains up to three times higher throughput than the normal program executed on the same cell. While FSP is not a novel idea for NAND flash, FSP is widely deployed in 3-D NAND SSDs [5], [6].

Wordline-based FSP operations are slated to effectively shorten program latency and improve the throughput of written data, but FSP does not account for the impact of differently typed pages on read latency. Multibit flash sets multiple read reference voltages to read differently typed pages in a word line [7]. The reference voltage divides a cell into multiple states, and the query time of each state is inconsistent.

Manuscript received 25 December 2022; revised 14 April 2023 and 10 June 2023; accepted 8 July 2023. Date of publication 11 July 2023; date of current version 22 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62072214, and in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021B1515120048. This article was recommended by Associate Editor P. Gupta. (Corresponding author: Yuhui Deng.)

Shujie Pang, Yuhui Deng, Zhaorui Wu, Genxiong Zhang, and Jie Li are with the Department of Computer Science, Jinan University, Guangzhou 510632, China (e-mail: p_shujie@163.com; tyhdeng@jnu.edu.cn; diom_wu@163.com; zgx787839110@163.com; lijiegxmd11@163.com).

Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36830 USA (e-mail: xqin@auburn.edu).

Digital Object Identifier 10.1109/TCAD.2023.3294452

Since a page needs to find a specified state when data are retrieved, the differently typed pages exhibit various read latencies. Wu et al. [8] revealed that LSB and CSB pages of TLC flash have the lowest 66 μ s and highest 88 μ s read latency, respectively. Liu et al. [1] studied the 3-D TLC NAND flash developed by Toshiba and measure the read latency of different pages as 62 μ s (LSB), 79 μ s (CSB), and 59 μ s (MSB). Besides, some developers also employ the different read latencies to evaluate their strategies [3], [9]. Unfortunately, this phenomenon inevitably affects the SSD read performance. Specifically, if hot read data are placed on high-latency pages, the read latency of flash memory will increase significantly. Additionally, read requests are usually dominated by small requests, and the size of most read requests is smaller than 32 kB [10]. However, in 3-D TLC NAND flash, each word line has three pages, and the size of a page is usually 16 kB, which means that a word line can store 48 kB of data. Therefore, it is an extremely low probability that all three pages on a word line are hot-read pages. At this point, 3-D TLC flash presents a new demanding requirement for data allocation strategies: *placing hot data on low-latency pages*.

Flash manufacturers implement multiplane operations to optimize the internal parallelism of NAND flash, thereby boosting SSD performance. The multiplane operation controls multiple planes of a flash chip to execute parallel commands. According to command types, the multiplane operation is divided into two commands: 1) multiplane program (MPP) and 2) multiplane read (MPR). A combination of MPP and FSP is undoubtedly expected to maximize write throughput. In TLC flash memory, since MPP requires more than two planes to support MPP, the FSP+MPP operation can program at least six pages in a program latency. Regrettably, FSP is not friendly to the read performance of MPR. MPR improves read throughput by reading consecutive pages stored in different planes, but FSP reduces the parallelism of read requests by placing consecutive pages on the same word line, enlarging read latency. Therefore, the combination of FSP and multiplane operations imposes another prerequisites for data allocation strategies: *ensuring parallelism of read requests*.

Although current SSDs are equipped with DRAM to ensure read/write latency, DRAM is usually used to serve write requests to enhance the lifetime and reliability of SSDs. There are three main reasons as follows. First, as the bits per cell in 3-D NAND flash increases, multibit flash-based SSDs have an extremely limited number of program/erase cycles. To improve the lifetime of SSDs, DRAM primarily serves write requests to trim the size of data written to flash memory. Second, since the program latency of flash memory is much higher than the read latency, hot-write data placed in DRAM can effectively improve flash performance. Consequently, an increasing number of DRAM optimization researches focus on improving the service efficiency of DRAM for write requests, while ignoring the support for read requests [11], [12], [13]. Finally, it is worth noting that the cached data in DRAM usually has strong temporal locality. However, the temporal locality of read requests is poor [10]. Accordingly, DRAM—having limited support for read requests—fails to address the impact of FSP+MPP and uneven page latency on read latency. Unlike write requests with uncertain physical addresses, the physical

addresses accessed by read requests are specified beforehand. For this reason, data placement location has an immense impact on read requests. We argue that effective data allocation schemes have a strong likelihood to alleviate the negative influence of FSP+MPP and uneven latency on read latency. We demonstrate in this study that judicious data allocations facilitate substantial improvement in data throughput—promoting 3-D NAND flash performance.

Motivated by the above observations and analysis, we propose FSPDA, an FSP data allocation scheme, to revamp read throughput of 3-D NAND flash. In an effort to be compatible with FSP operations, FSPDA sets every three pages as a queue to be assigned to specified planes. FSPDA allocates logical pages to different queues according to page distance, and a cross-allocation method is employed to assign consecutive pages with a purpose of maintaining data locality and improving read parallelism. Moreover, the novel scheme allots three pages in a queue to physical pages with different latencies according to cache read hits of the pages, thereby placing hot read data on low-latency physical pages to speed up SSD performance. We shed bright light on the implementation details of TLC-based FSPDA. It is worth noting that FSPDA is applied to any multibit 3-D NAND flash-based SSDs. Our contributions are summarized as follows.

- 1) We present two intriguing observations. First, FSP+MPP operations designate consecutive logical pages to the same word line, which adversely affects read parallelism and latency of the consecutive pages. Second, data with high read hits have a high read frequency after being written back to flash memory, and thus hot read data can be confirmed by read hits.
- 2) We develop an FSP data allocation scheme FSPDA that rationally allocates the physical addresses of logical pages in accordance with the logical distance and read hits of the pages. FSPDA effectively advances the read parallelism and curbs hot read latency, thereby boosting multibit 3-D NAND flash performance.
- 3) We demonstrate the superiority of FSPDA over the other competitors in terms of the number of MPR counts, read access counts, and average read/write response time under eight real-world workloads.

The remainder of this article is organized as follows. Related work and background are presented in Sections II and III, respectively. Section IV discusses the motivations of FSPDA. We elaborate on the design of FSPDA in Section V. The performance evaluation is presented in Section VI. Finally, Section VII concludes this article with comments.

II. RELATED WORK

3-D NAND flash-based SSD is one of the active research directions in the realm of storage systems, where performance and stability issues have been intensively investigated by researchers and institutions.

A. Data Allocation

Data allocation strategies, as a main factor to determine the data physical addresses, receive a lot of attention from the industry and academia. There are static and dynamic data allocation strategies in existing works [14].

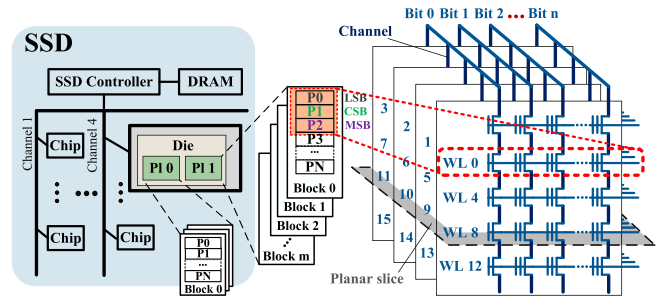
For instance, Hu et al. [15] evaluate dynamic and static allocation strategies, pointing out that the dynamic strategy outperforms the static in terms of write and wear leveling performance. Reddy et al. [16] designed a hybrid data distribution method powered by the multilevel parallel characteristics of SSDs, which orchestrates dynamic or static allocation for various parallel layers of SSDs. Furthermore, a 3-D NAND flash requires new page allocation algorithms to address the 3-D NAND-specific problems, including the additional disturbance from the vertical and horizontal directions [17], [18], [19]. Chang et al. [18] proposed a page allocation scheme to program written data to word lines with no programmed word lines around, thereby mitigating program disturb. Wang et al. [19] advocated for evenly distributing data into different physical blocks to alleviate bit errors caused by data hotspots. Unfortunately, these aforementioned allocation schemes only consider the 3-D NAND SSD architecture without being aware of the performance impact of the advanced operation commands issued by 3-D NAND SSDs. In contrast, our FSPDA not only takes into account the 3-D NAND architecture but also utilizes various advanced SSD operations to bolster the I/O performance of SSDs. Meanwhile, FSPDA is adroit at optimizing the overall SSD performance without excessively consuming SSD lifetime.

B. Advanced Operations

To reshape the performance of 3-D NAND flash-based SSDs, SSD manufacturers have developed advanced operations, such as multiplane operation and FSP. The multiplane and FSP operation, as advanced approaches to improving SSD throughput, are widely concerned by many scholars and SSD manufacturers. For example, Shahidi et al. [20] and Wang and Xie [21] implemented plane-level parallel garbage collection strategies by the virtue of multiplane operations. These parallel strategies, performing garbage collection inside the plane, avert channel occupation by data transfer during garbage collection. However, the advanced operations have adverse impacts on SSD performance. To alleviate page waste generated by MPP and FSP, Liu et al. [22] designed a resource allocation scheme (GSSA) customized for 3-D NAND SSDs. The scheme utilizes FSP and MPP with multiple granularities to render customized program services for write requests. GSSA—enhancing the space utilization of flash memory—does not solve the problem of increased read latency incurred by the advanced operations. Wu et al. [23] proposed an FSP-based parallel data placement strategy to maximize the read parallelism of continuous data, thereby shortening the read latency of SSDs. Unfortunately, the above studies ignore the latency discrepancies of physical pages in a multibit flash. In contrast, our scheme employs FSP+MPP to place hot data in the most appropriate physical location, aiming to curtail the read latency of hot read data.

C. Latency Optimization

Latency is one of the main metrics for evaluating the performance of a storage device. From the perspective of users, lowering data read latency is desirable. Currently, there have been a raft of works on the in-depth study of latency optimization. For instance, Yan et al. [24] proposed TTFLASH



cell in the word lines stores up to 3-bit information, and in this case, such a cell is referred to as triple-level cell or TLC. A TLC word line is accessed as three separate pages: 1) LSB; 2) CSB; and 3) MSB pages (see, for example, pages P0, P1, and P2). The NAND flash, being able to execute only one operation at a time, furnishes three basic operations: 1) program; 2) read; and 3) erase operations. The traditional program and read operations are in a unit of page, while an erase operation is performed in a unit of block. Conventionally, the three differently typed pages within a TLC word line are programmed separately page-by-page. Meanwhile, prior studies revealed that the three types of pages have significantly diverse program latencies [28], [29].

FSP: To improve write throughput, SSD manufacturers implement FSP operations to expand the range of program operations [5], [6]. The tradition program operation only program one page out of the LSB, CSB, and MSB pages in one of the planes, and the pages must be programmed in order from low to high. Unlike the traditional program operation with a unit of page, the FSP operation is performed in a unit of word line, which will program all pages of a word line in a program latency. Meanwhile, the WL addresses of programming should be arranged in an ascending order within a block. In the MLC flash, FSP programs LSB and MSB pages of a word line in a program latency [30]. Likewise, in the TLC flash, FSP can program LSB, CSB, and MSB pages of a word line in a program latency [23] (see Fig. 1, for example, pages P0, P1, and P2), thereby providing higher write throughput. Especially, in the mainstream flash memory, the page size is 16 kB, and the traditional program operation writes 16 kB of data in a program latency. However, in the FSP-based TLC flash, FSP can write 48 kB of data in a program latency, which provides $3 \times$ write throughput.

Multiplane Operation: The multiplane operation—performing program/read operations on multiple planes at the same time—is divided into two commands: 1) MPP and 2) MPR. Multiplane operations theoretically provide at least $2 \times$ throughput. Take Fig. 1 as an example, the flash memory employs MPP/MPR operations to simultaneously read/program pages P1 in planes Pl 0 and Pl 1, respectively. In particular, in the TLC flash with two planes, FSP+MPP can program up to six pages in a program latency. For instance, pages P0, P1, and P2 in plane Pl 0 can be programmed at the same time as pages P0, P1, and P2 in plane Pl 1. Unfortunately, FSP+MPP imposes a negative impact on MPR, and we shed bright time on this issue in Section IV-A. Please note that we only employ TLC flash as an example in this article to introduce the impact of FSP+MPP on SSD performance. However, we must clarify that FSP+MPP is not limited by the flash type, which can work on any type of flash memory. For example, Cernea et al. [30] implemented FSP operations in MLC flash to program all pages on a word line. Moreover, Wu et al. [23] and Liu et al. [22] proposed SSD optimization strategies on FSP+MPP-based TLC and FSP+MPP-based QLC flash.

B. Read Operation in Multibit Flash Memory

According to the amount of charge stored, an NAND flash cell stashes multibit data by dividing the threshold voltage

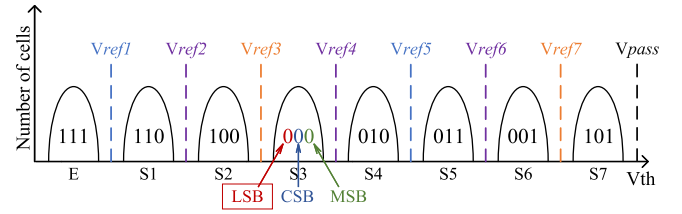


Fig. 2. State diagram and V_{th} distribution of a TLC flash cell.

(V_{th}) into multiple regions. A region to which the V_{th} of a cell belongs represents data storage value. Recall that a TLC cell stores three bits, and the threshold voltage of the cell is divided into eight states (see E, S1, S2, ..., and S7 in Fig. 2). If no charge is present, the cell is the erased state E (111). As illustrated in Fig. 2, seven read reference voltages (V_{ref}), V_{ref1} to V_{ref7} , divide a TLC cell into eight states, E and S1 to S7, which are decoded as state values 111, 110, 100, 000, 010, 011, 001, and 101, respectively. Specifically, the state values are, from left to right, the values of LSB, CSB, and MSB.

During a read operation, all slices exclusive of a target page are applied with a read pass voltage (V_{pass}), which ensures that the target page in the selected word line is read. Amid a target page reading procedure, a read operation is obtaining data from the page: only the target bit of information is extracted from each cell of a selected word line. Flash memory uses a series V_{ref} to retrieve the value of the target bit. According to the type of the target page, read reference voltages are divided into three sets, namely, LSB page $\rightarrow V_{ref3}$, V_{ref7} , CSB page $\rightarrow V_{ref2}$, V_{ref4} , V_{ref6} , and MSB page $\rightarrow V_{ref1}$, V_{ref5} . As only a single-read voltage level can be tested at a time, the read operation is accomplished in an iterative process. For example, if an LSB page is read, flash memory will check V_{ref3} and V_{ref7} one by one to determine whether the state value of the LSB is 1 or 0. The page latency is determined by the number of read reference voltages for checking its state value. Therefore, LSB and MSB pages in a 3-D TLC NAND flash typically have low read latency, whereas CSB pages are inclined to be high-latency pages [1], [8].

IV. MOTIVATION

In this section, we first analyze the impact of FSP+MPP on the MPR read performance. Then, we experimentally examine the relationship between read hits in cache and hot/cold data in flash memory, thereby rationally allocating the physical location of hot/cold data.

A. Impact of FSP+MPP

In multibit 3-D NAND flash-based SSDs, multiplane, and FSP operations are adopted to smooth the uneven program latencies of pages and to increase I/O throughput. Unfortunately, FSP+MPP operations heavily downgrade the parallelism of consecutive data, which in turn deteriorates the read efficiency of MPR operations. To gain insight into this issue, we analyze the negative impact of FSP+MPP-based data allocation on reading consecutive data.

A combination of the traditional program and MPP operations—in 3-D TLC NAND flash—can distribute consecutive logical pages of write requests to different planes.

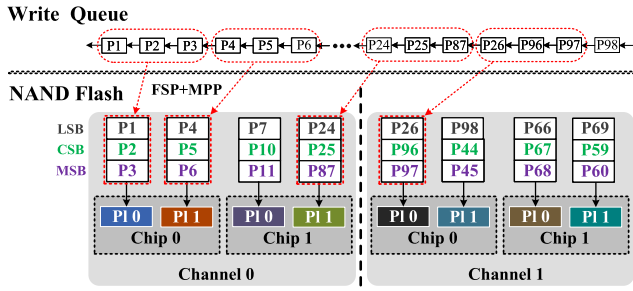


Fig. 3. Impact of FSP+MPP on MPR.

Specifically, consecutive pages $n - 1$, n , and $n + 1$ of a write request are allotted to three different planes, increasing write throughput. Since consecutive pages are distributed on different planes, MPR operations offer high parallelism and large read throughput. After flash manufacturers developed FSP operations, unfortunately, the combination of FSP+MPP instead degrades the high parallelism of read operations provided by MPR. While FSP smooths the program latencies between different pages and increases the number of pages programmed in one shot, FSP places some consecutive data on the same plane, which leads to unacceptable performance of MPR. Therefore, FSP+MPP fosters the continuity of local data while reducing the parallelism of read requests.

Let us take Fig. 3 as an example to reveal the impact of FSP+MPP operation on MPR in TLC flash memory. Fig. 3 sketches that the data in the write request queue, divided into three queued pages as a group, are written to TLC NAND flash by FSP. Because an SSD supports multiple levels of parallelism (channel, chip, and MPP), the planes in different channels or chips can also perform read/write operations in parallel like MPP; that is, the eight planes in Fig. 3 simultaneously perform read and write operations. Thus, the queued pages are allocated to all the eight planes to guarantee the operational advantages of FSP+MPP, thereby maintaining the maximum write throughput of flash memory. It is noteworthy that this FSP+MPP distribution is extremely unfriendly to MPR. Specifically, flash memory utilizes FSP operations to place three consecutive pages on one-word line, which cannot be read simultaneously by MPR. For example, pages P1, P2, and P3 are placed on the same word line of the same plane. Thus, when a read request accesses the three pages, the flash memory can only take the $3 \times$ read latency to sequentially retrieve the pages.

Furthermore, to verify the negative impact of FSP on read parallelism, we investigate the impact of FSP on MPR counts. We undertake this group of experiments on the eight workloads with respect to two SSDs with FSP and without FSP. To achieve fair comparisons, we normalize the experimental results of different SSDs to the SSD without FSP. The experimental parameters are shown in Table I in Section VI-A. The experimental results are plotted in Fig. 4, where more MPR counts indicate better read parallelism. In Fig. 4, the MPR counts of the SSD with FSP are significantly lower than that of the SSD without FSP. Therefore, FSP+MPP will degrade the read performance of MPR if both FSP and multiplane operations are applied to flash memory.

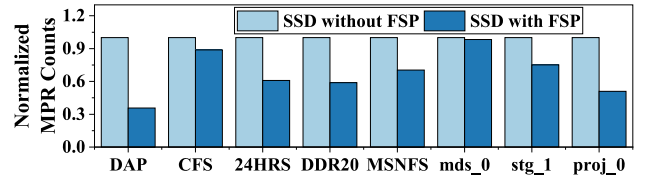


Fig. 4. MPR counts of different program operations.

There exists a strategy that allocates queued pages on different planes of flash memory according to logical distance [23]. Only if the distance between two queued pages is greater than the specified distance, the pages will be placed on the same plane. Regrettably, this allocation strategy places consecutive pages in different parallel units indefinitely, which is unsuitable for modern SSDs and workloads. Again, we take Fig. 3 as an example: this strategy unwisely designates pages P25 and P26 to different channels. Evidence shows that channel resources are extremely precious in SSDs. Garbage collection, as an integral part of SSDs, occupies channels to migrate valid data, and the GC-blocked channel enlarges read and write latency. Unfortunately, the strategy will allocate consecutive pages on different channels (such as P25 and P26 in Fig. 3): read requests corresponding to the consecutive pages may suffer the $2 \times$ garbage collection probability. In addition, Liu et al. [10] tested 600 workloads to unfold that most read requests are small requests, the size of which normally is smaller than 32 kB. Specifically, eighty percent of the read requests are less than 32 kB in nearly eighty percent of the workloads. Because the minimum physical page supported by SSDs is 16 kB, a read request only needs to be serviced by one or two pages in most cases. Therefore, the strategy, maximizing the read parallelism of consecutive data, is unnecessary in practice.

B. Correlation Between Cache Hit and Flash Read

Recall that (see Section III-B), more often than not, physical pages in multibit NAND flash have different read latencies depending on the distribution of reference voltages. Therefore, placing hot read data on low-latency physical pages is undoubtedly the most appropriate choice for bolstering read performance. Due to the randomness of read requests, however, it is arduous to determine whether flash data are hot or cold before being accessed by read requests. Fortunately, a cache that primarily serves write requests is introduced in high-performance SSDs to improve performance and lifetime. Although the cache furnishes a limited read performance improvement, caching techniques enable a number of read accesses to swiftly access cached data, which can be used to verify hot and cold data.

To give insight into the relationship between read hits in cache and hot/cold data in flash memory, we conduct five experiments to delve into the read access frequency of pages with different read hits in flash memory. Among the experiments, the read hits refer to the counts of reading a cached logical page, and the read access frequency indicates the frequency of reading the specified page in flash memory. More specifically, we investigate in the five experiments the read access frequency of pages with read hits = 0, read hits > 0,

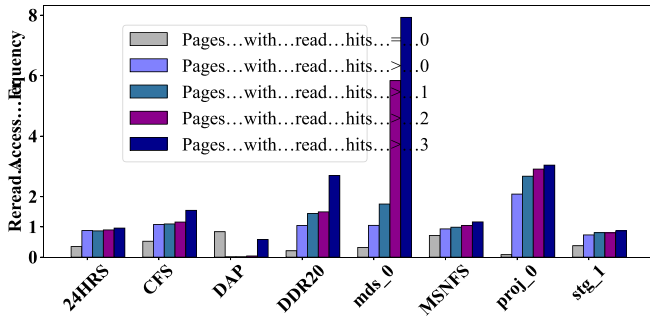


Fig. 5. Correlation between cached read hits and read frequency.

read hits > 1 , read hits > 2 , and read hits > 3 , respectively. All the experiments detail are carried out on the SSDSim [15] simulator driven by the eight traces (see Section VI-A for details).

Fig. 5 shows the read access frequency of pages in five cache read hit states on the eight workloads. For example, “Page with read hits = 0” reveals that the read hits of a logical page in the cache is 0, and the bar graph represents the read access frequency of reading the logical page in flash memory. In Fig. 5, “Page with read hits = 0” has the lowest read access frequency except for *DAP*, and the read access frequency is less than 1 under entire traces. The lower frequency entails that after a cached logical page with 0 hits is written back to a flash memory, the probability of reading the logical page is very slim in the flash memory. We also observe from Fig. 5 that as the read hits of logical pages climbs, its read access frequency also rises. Obviously, the cache read hits are positively correlated with the read access frequency. If a cached logical page with high hits is written back to flash, the logical page still has a high chance of being retrieved in the flash. For this reason, we adopt the cache hits to discern hot read data from cold ones, thereby optimizing data allocation in SSDs.

V. FSPDA SCHEME

Since SSD manufacturers have developed a new program method FSP for multibit 3-D NAND flash, traditional data allocation strategies are incompatible with FSP operations. Therefore, the performance degradation caused by the FSP operation can easily offset such a performance improvement of the traditional strategies. As discussed in Section IV, the existing FSP+MPP allocation schemes fall short on satisfying the high concurrent read/write of consecutive data. To achieve the full benefits from the high read and write throughput brought by state-of-the-art 3-D TLC NANDs, we propose a novel FSP data allocation scheme referred to as FSPDA.

We start this section by presenting an overview of our FSPDA, followed by the detailed implementation of FSPDA, which consists of a data queuing unit and a page matching module fueled by hot/cold cached data.

A. System Overview of FSPDA

Due to the limitations of FSP+MPP, 3-D TLC NAND flash-based SSDs suffer from excessive waste of read parallel resources and the problems of uneven page latency. To revamp the read performance of 3-D NAND flash-based SSDs, we advocate for an FSP data allocation scheme –FSPDA. Our

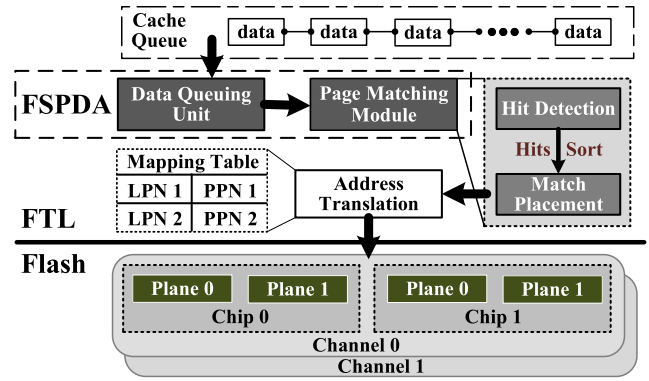


Fig. 6. Overview of FSPDA for 3-D TLC NAND flash.

proposed FSPDA scheme differentiates itself from existing schemes by curtailing the waste of read parallel resources and rationally distributing hot and cold data.

FSPDA strives to optimize the I/O efficiency of SSDs by adjusting data allocation amid writing cached data to flash. To efficiently optimize the performance and lifetime of SSDs, we make SSDs cache all write requests in a cache queue. When the cache queue is full, FSPDA allocates several data write-back queues to write cached data back to flash memories, where the number of the write-back queues is the same as those of planes in flash. During the course of writing back, FSPDA dynamically distributes cached logical pages on multiple planes according to the distance of logical pages, and physical page addresses are allocated to the logical pages based on cache hits: FSPDA aims at reducing the read latency of 3-D TLC NAND flash.

Fig. 6 depicts the overall system design of our proposed FSPDA, which is deployed in the SSD controller to orchestrate data allocations. The overarching goal of FSPDA is to shorten request latency through judicious data allocation scheduling. In particular, FSP+MPP-based 3-D TLC SSDs have two requirements for data allocation strategy (see Section I): 1) placing hot data on low-latency pages and 2) ensuring parallelism of read requests. To this end, FSPDA makes use of two-stage data scheduling to strengthen the parallelism of read requests and slash the hot read latency. Specifically, the two-stage data scheduling is implemented by the two core modules of FSPDA, namely, the data queuing unit and the page matching module. According to the addresses of logical pages, the data queuing unit is responsible for distributing consecutive pages to different parallel units while maintaining a certain data locality. The page matching module is in charge of matching the mapping between logical pages and physical addresses by cache hits. Below, we elaborate on the data queuing unit and the page matching module in Sections V-B and V-C, respectively.

B. Data Queuing Unit

To upgrade read parallelism and maintain data locality, FSPDA coordinates the data queuing unit. In what follows, we first explain the implementation details of the data queuing unit, and then we shed light on an example to illustrate its practical role.

Our FSPDA manages write-back queues to temporarily store cached logical pages for writing back to flash, and the data queue unit is designated to place logical pages to the specific queues during the write-back process. Specifically, FSPDA assigns a write-back queue to each plane in an SSD, and it is stipulated that pages in a write-back queue can only be written to its corresponding plane. In addition, when it comes to a single FSP operation, the length of the write-back queue is the maximum number of programmable pages—the number of physical pages owned by a word line. Taking a TLC NAND as an example, because each word line contains three pages, the FSP operation can program three pages: the length of the write-back queue is three.

When the cache space is full, SSDs write cached pages back to flash for servicing incoming write requests. To ensure high read parallelism and data locality, the data queue unit of FSPDA allots the cached pages to multiple write-back queues according to the logical addresses of the cache pages. More prosaically, the data queuing unit mainly considers three factors to make wise allocation decisions. First, recent evident [10] suggests that the size of most read requests is less than 32 kB, and the physical page size is 16 kB in the current 3-D NAND flash. Most read requests can be satisfied by reading two physical pages in parallel. Second, garbage collection—GC—occupies channel and chip resources, which in turn affect the completion time of parallel read operations. To alleviate tail latency induced by GC, the parallel read operations across multiple channels must be minimized. Third, scattered consecutive pages with high parallelism have extremely poor data locality, and data locality directly impacts garbage collection efficiency. Therefore, our FSPDA, aiming to enhance read performance and data locality, divides the write-back queues of the same chip into a group, where the data queue unit adopts the FSP+MPP operation to store consecutive logical pages by the cross-allocation method within the group. It is worth noting that the number of pages per group is the maximum number of pages programmed by the FSP+MPP operation. Since FSP+MPP performs FSP operations on multiple planes simultaneously, and the FSP operation programs all pages on a word line, the maximum number of pages programmed by FSP+MPP is the number of pages per word line multiplied by the number of planes per chip. Specifically, the number of pages per group is equal to the product of the number of pages per word line and the number of planes per chip.

The implementation details of the data queuing unit in FSPDA are articulated as follows. In the data queuing unit, there are five parameters D , D_m , R_T , R_C , and R_A that determine the physical locations of cached pages, and the units of the parameters are page. Among them, D is the logical distance (the number of logical pages) between a cached page and a queued page, and D_m is the minimum distance. R_T , R_C , and R_A represent the remaining space of a target group, the remaining space of all groups in the channel of the group, and the remaining space of all groups in an SSD, respectively. In particular, the remaining space of a group is equal to the number of pages per group minus the number of queued pages in the group. More specifically, R_T is the remaining number of logical pages that can be stored in a target group; R_C is the remaining number of logical pages that can be stored in the groups of the channel where the target group is located; R_A

is the remaining number of logical pages that can be stored in all groups. When the insufficiency of cache space triggers write-back operations, cached pages are written back to flash memories sequentially from the cache queue tail. If all the write-back queues are empty, the cached page is directly added to the write-back queue with the least write-back counts to strike a good write balance. Conversely, if there is a nonempty queue, FSPDA checks the distance D between a cached page and queued pages, followed by assigning the pages to the specified queue. In particular, the data queuing unit classifies page allocations into four cases, the priorities of which are decreasing from 1) to 4).

- 1) If a queued page has the minimum distance D_m in all queued pages, and D_m is no greater than R_T , the cached page is directly inserted to the group of the page (target group).
- 2) If a page has the minimum distance, where $R_T < D_m \leq R_C$, the cached page is designated to other groups of the same channel.
- 3) If a page satisfies $R_C < D_m \leq R_A$, the cached page is appointed to other channels.
- 4) In case of $D_m > R_A$, the cached page is allocated by the write-back counts.

The parameters R_T , R_C , and R_A can determine whether pages are placed on the same chip or channel, which is very important. Chips and channels are important components in SSDs. Undoubtedly, placing consecutive pages on a chip significantly enhances data locality compared with placing them on a channel. Similarly, compared with multiple channels, consecutive pages placed in a single channel exhibit stronger data locality, and mitigate the impact of GC-blocked channels. Therefore, FSPDA employs the parameters R_T , R_C , and R_A to place pages on different chips or channels according to logical distance, thereby improving data locality.

Furthermore, a distinctive trait of the data queuing unit lies in the *cross-allocation method*. The cross-allocation method places consecutive pages on different queues, thereby improving the read parallelism of the consecutive pages. Specifically, consecutive pages in a group are allocated sequentially to each write-back queue of the group by the cross-allocation method. Especially, when the number of consecutive pages is greater than the number of write-back queues per group, the remaining pages are reallocated from the first queue. For example, in a flash chip with three planes, five consecutive pages P1, P2, P3, P4, and P5 are allocated to a group with three write-back queues Q0, Q1, and Q2. First, pages P1, P2, and P3 are assigned to Q0, Q1, and Q2, respectively. Then, the remaining pages P4 and P5 are also allocated starting from Q0 and placed on Q0 and Q1, respectively. In this case, FSPDA significantly boost the read parallelism of consecutive pages while alleviating the adverse impact of GC-blocked channels. Meanwhile, because the data queuing unit allocates logical pages with close addresses to the same chip or channel, FSPDA is slated to maintain high data locality, which leads to optimized efficiency of garbage collection.

In the following TLC flash example, we demonstrate that our FSPDA is adroit at immensely promoting read parallelism and data locality. Let us assume that the cache queue is full and frees up space for new write requests issued by the host. Cached pages P1, P2, P3, ..., P96, etc. are written back to

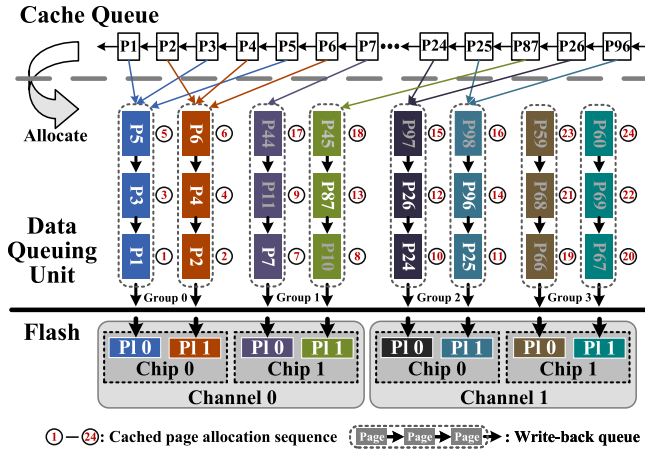


Fig. 7. Data queuing unit workflow.

flash memories in a sequence way, and the initial write-back queues are empty (see Fig. 7). First, FSPDA allots consecutive logical pages P1–P6 to Group 0 ($D_m \leq R_T$). Meanwhile, according to the cross-allocation method, the six pages P1–P6 with $D = 1$ are, respectively, distributed on different queues. At this point, any two consecutive pages are read simultaneously. Next, although the distance D between the cache page P7 and the queue page P6 is 1, P7 is placed on Group 1 because Group 0 is full ($R_T = 0 < D_m \leq R_C = 6$). Subsequently, P10 and P11 are assigned in Group 1 ($D_m \leq R_T$). P24 is appointed to Group 2 in channel 1 ($R_C < D_m \leq R_A$), and thus P25 and P26 are also placed on the group. Then, since P87 is far away from the previous pages ($D_m > R_A$), it is allocated to Group 1 because the write-back counts are 0. Likewise, P96–P98 are alternately assigned to Group 2. Therefore, in the 3-D TLC NAND flash with two planes per chip, FSPDA can impressively boost the read parallelism of two consecutive pages. In the above example, it is worth noting that in Fig. 7, we omit the allocation description of some pages. There are two primary reasons. First, since the cache queue in Fig. 7 is too long, we replace some cached pages with ellipses and directly indicate the allocated positions of the cached pages in the write-back queues after being allocated. Second, we do not explain the allocation process of some pages (e.g., P44, P45, P66, etc.) in this article, because the pages have a similar allocation process to the pages explained in this article. For example, the allocation process of P44 and P87 is similar ($D_m > R_A$). After P44 is allocated, the allocation process of P45 is similar to that of P2 ($D_m \leq R_T$). Moreover, we observe from Fig. 7 that logical pages with close addresses are usually allocated to the same chip or channel, which effectively alleviates the weakening of data locality incurred by the parallelism improvement.

C. Page Matching Module

The data queuing unit of FSPDA mitigates the degradation of read parallelism introduced by FSP+MPP operations. Nevertheless, there are still opening problems with FSP operations. Although FSP smooths program latencies, there exists uneven read latency in multibit flash, which disturbs the read latency for hot data. As such, FSPDA implements a page matching module to allocate the physical locations of

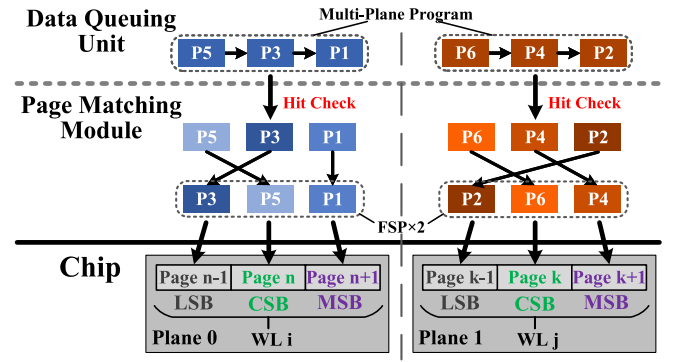


Fig. 8. Page matching module workflow.

hot/cold pages, thereby curbing the hot read latency. In what follows, we explain the implementation details of the page-matching module, and we illustrate the advantages of the module through a concrete example.

To optimize performance and enlarge lifetime, high-performance SSDs usually adopt cache to suppress the number of read and write requests served by NAND flash. Due to the limited program times for a NAND cell, the SSD cache is inclined to serve write requests, thus extending the SSD lifetime. Importantly, the latest TLC and QLC NAND flash only provide a few thousand and hundreds of program guarantees per cell, respectively. Accordingly, to cache more hot write data, the remaining data will be written back to flash regardless of the read heat. Moreover, because read requests are mostly less than 32 kB (one or two pages), the three pages programmed by a single FSP operation in TLC NAND may have different hot and cold properties in the cache. Meanwhile, it is noteworthy that hot read data in cache usually has a high read frequency in flash (see Section IV-B). Thus, FSPDA develops the page matching module to pair hot-read cached pages with low-latency physical pages, expecting to cut back the latency of hot-read requests.

The implementation details of the page matching module in FSPDA are articulated as follows. When the data queuing unit is completed, the page matching module assigns logical pages per write-back queue to different physical pages in a word line according to their cache read hits. Initially, the page matching module checks the cache read hits of queued pages per queue, and the module dictates the hot/cold properties of the pages by the read hits. Next, according to the read voltage distribution of multibit NAND flash, the page matching module determines the read latency of physical pages on a word line. Finally, the queued pages with high hits are appointed to low-latency physical pages, and the queued pages with low hits are placed on high-latency pages. In doing so, FSPDA makes full use of various latencies to enhance read efficiency.

Taking Fig. 8 as an example, we demonstrate how our FSPDA employs the page matching module to efficiently matches logical and physical pages. Fig. 8 sketches that the queued pages of Group 0 in Fig. 7 adopt the page matching module to elect physical pages in three steps. First, FSPDA checks the read hits of pages (P1, P3, P5) and (P2, P4, P6) in two write-back queues, respectively. The pages with darker colors have more read hits (see Fig. 8). Then, FSPDA detects the addresses and latency of the physical page. In Fig. 8, we

take a 3-D TLC NAND flash (BiCS) produced by Toshiba as an example. In a word line of the flash, LSB and MSB pages usually have the lowest read latency, whereas CSB has higher latency (see Section III-B). Since MSB pages in the multi-bit flash usually have more bit errors, FSPDA allocates P2 and P3 with the highest hits to LSB pages $n - 1$ and $k - 1$, respectively. Meanwhile, pages P5 and P6 with the lowest hits are, respectively, allocated to CSB pages n and k , and the remaining pages P1 and P4 are allocated to MSB pages. Finally, the two queues with allocated physical addresses are written to the flash memory through FSP+MPP operations. Fig. 8 illustrates that FSPDA can effectively place hot data in low-latency physical pages, thereby significantly improving the read performance of flash memory.

In a nutshell, FSPDA invokes the flexible data allocation methods to alleviate the read performance degradation introduced by FSP operations: FSPDA regulates the plane addresses of logical pages by calculating the distance of pages. Leveraging the parallelism mechanism in flash-based SSDs, FSPDA places consecutive pages in different parallel units while maintaining good data locality. Then, FSPDA assigns the logical pages with higher hits to the physical pages with lower latency. In doing so, FSPDA not only provides high concurrency for a majority of read requests but also warrant low read latency for hot read requests. Consequently, FSPDS dramatically optimize SSD performance.

D. Discussion

In this section, we first analyze the relation between the write-back queue size and DRAM performance and then analyze the impact of DRAM size on FSPDA.

1) *Write-Back Queues Size*: FSPDA allocates the physical locations of logical pages by setting multiple write-back queues, which reduces read latency and maintains certain data locality. Fortunately, the space occupied by the write-back queues is extremely limited and small. Specifically, in FSPDA, the number of write-back queues is equal to the number of planes in an SSD, and the length of each queue is the number of pages per word line. For example, assuming an SSD has N planes and each word line has L pages, the size of the write-back queues should be $N \times L \times P_s$. P_s represents the size of a page. We take the TLC flash-based SSD adopted by the evaluation section as an example. The SSD has eight planes and its structure is consistent with Fig. 7. In the SSD, the DRAM size is 128 MB and the page size is 16 kB. At this point, the size of the write-back queues in the SSD is 384 kB (eight planes $\times$ three pages $\times$ 16 kB). It can be seen from the example that compared with the total DRAM size, the size of write-back queues in FSPDA is extremely small (less than 3% of the total size). Furthermore, compared with the hundreds of megabytes or several gigabytes of DRAM in the current high-performance SSDs, the write-back queues of FSPDA only occupy hundreds of kilobytes, which is negligible. The slight reduction of DRAM size has an extremely limited impact on its performance. Therefore, the write-back queues implemented by FSPDA will not affect DRAM performance.

2) *Impact of DRAM Size on FSPDA*: FSPDA implements the data queuing unit and the page matching module to allocate the physical locations of logical pages, thereby improving SSD

TABLE I
SIMULATION PARAMETERS

SSD parameters	
SSD Capacity	128 GB
(Channels, Chips)	(2, 4)
DRAM capacity	128 MB
(Blocks/plane, Pages/block)	(1888, 576)
(Page size, Cell density)	(16 KB, TLC)
Over provisioning ratio	0.25
Cleaning policy	Greedy
Latencies	
Page Program	900 μ s
Page read (LSB, CSB, MSB)	(66 μ s, 88 μ s, 66 μ s)
Block erase	10000 μ s

performance. Among them, the data queuing unit allocates logical pages with close logical distance to adjacent physical locations. Since the logical distance is not significantly related to DRAM size, the data queuing unit is usually not affected by DRAM size. Unfortunately, DRAM size will affect the page matching module. Specifically, the page matching module detects hot read data by cache read hits, and places the hot read data on low-latency physical pages. Therefore, the read-hit information of cached data is indispensable for the page-matching module. Regrettably, in a small DRAM, most cached data is written back to flash before servicing read requests. There is only a small number of read-hit information in the small DRAM, which is not friendly to the page-matching module. Contrarily, a large DRAM will hold more cached data, and the data can be cached longer in DRAM. Thus, the large DRAM can better distinguish the read hits of different data, and provide more read-hit information for the page-matching module. Fortunately, as the current SSD capacity increases, the DRAM size also increases rapidly. Since a large-capacity DRAM can provide more cached data and hit information, we believe that FSPDA will have an outstanding performance in the current SSDs.

VI. EVALUATION

A. Experimental Setups

To facilitate accurate performance analysis at the storage level, we adopt the SSDsim [15] simulator to implement and evaluate our FSPDA. SSDsim, as a trace-based public simulator, has been verified against real-world hardware platforms. Meanwhile, a wide adoption of SSDsim is found in a pile of studies investigating the performance of SSD-based storage systems. In our experiments, we simulate an SSD with a capacity of 128 GB, the configuration parameters of the simulated SSD are mainly derived from a prior study [10]: the evaluated SSD has four TLC 3-D NAND chips, each of which has 32 GB density. In addition, our experiments employ the read latency parameters reported in [8], which presents the performance of a representative 3-D CT NAND flash product—BiCS2 TLC from Toshiba. The detailed SSD parameters used in our empirical study are listed in Table I.

Workloads: We compare our FSPDA with a variety of schemes in terms of SSD performance on a total of eight

TABLE II
I/O CHARACTERISTICS OF EIGHT WORKLOADS

Trace	Total I/Os	Write Ratio	Avg_Write_Size	Avg_Read_Size
DAP	1086729	0.438	194.26	124.23
CFS	4477510	0.262	25.24	17.45
24HRS	5250996	0.817	23.32	212.22
DDR20	18195701	0.323	61.19	40.96
MSNFS	29345085	0.328	22.22	16.65
mds_0	1211034	0.881	14.47	47.42
stg_1	2196861	0.363	15.76	118.46
proj_0	4224524	0.875	81.41	35.67

realistic workloads. Among these eight workloads, workloads *mds_0*, *stg_1*, and *proj_0* are obtained from the revised SNIA traces [31]. The other five workloads are gleaned from the Microsoft Production Server [32]. Table II summarizes the workload characteristics of these eight traces. In particular, *24HRS*, *mds_0*, and *proj_0* are write-intensive workloads, whereas the other traces are read-intensive in nature. *Avg_Write_Size* and *Avg_Read_Size* signify the average size of write and read requests by the number of logical sectors, respectively.

Comparison Schemes: We compare FSPDA with three alternative solutions dynamic allocation, OSPADA [23], and SOML [10]. Additionally, we add the baseline+page-matching-module (Baseline+PMM) as the fifth scheme. The dynamic allocation in this study is serving as a baseline solution for all the other schemes. Since the scheme optimization is related to the access characteristics of traces, the same metric may have a significant performance gap on different traces. For fair comparison purpose, we normalize the experimental data of the tested schemes to the level of the dynamic allocation scheme. Moreover, we directly download the source code of OSPADA to be executed on SSDsim, and we simulate the SOML scheme on SSDsim. The state-of-the-art solutions, including the baseline (dynamic allocation), OSPADA, and SOML, are summarized as follows.

- 1) **Baseline:** The dynamic allocation scheme, a basic data distribution scheme, is implemented in SSDsim. The dynamic allocation technique adopts FSP+MPP operations to write pages to flash sequentially according to the cache queue order. Unfortunately, the baseline solution offered by SSDsim ignores the impact of read parallelism and uneven latency on SSD performance.
- 2) **OSPADA:** OSPADA is designed to maximize read parallelism for contiguous data: OSPADA reorders written data to distribute logically sequential data into different parallel units using the distance-aware round-robin strategy. Though OSPADA can significantly lift read efficiency, OSPADA magnifies the impact of garbage collection on I/O latency, and OSPADA overlooks the impact of different read latencies.
- 3) **SOML:** The SOML read operation implements the fine-grained parallel read operation in flash chips. SOML combines multiple small read requests in a plane by adding extra control circuits, thereby effectively improving read throughput. Unfortunately, the extra circuits occupy limited physical space accompanied by enlarged read latency.

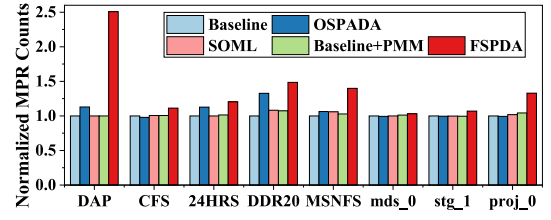


Fig. 9. MPR counts of all the evaluated schemes with eight traces.

Metrics: Metrics are of paramount importance for evaluating enhancements in NAND flash performance. In this study, we gauge system performance by leveraging six metrics: 1) MPR; 2) read access counts; 3) GC counts; 4) migrated pages; 5) response time; and 6) write amplification. The rationale behind the selection of these metrics is given as follows. First, MPR is a compelling indicator to assess the impact of a diversity of strategies on read parallelism. Second, read access counts reveals the number of read request accesses for different physical pages, and thus this metric analyzes the effect of FSPDA on low-latency and high-latency pages. Third, each block, in NAND flash memory, has a constraint on the maximum number of erase operations. Since each GC involves a block erase operation, GC counts intuitively express SSD lifetime. Fourth, migrated pages obviously pinpoint valid pages that are moved out of a victim block during the course of GC. Fifth, response time measures the time spent processing requests in a NAND flash-based SSD. Finally, write amplification represents the ratio between the number of actual write accesses and the number of write I/Os issued by a host.

B. Performance Evaluation

1) **MPR:** In this group of experiments, we pay attention to MPR for evaluating the scheme performance. Specifically, we compare our FSPDA with the other four alternative solutions—Baseline, OSPADA, SOML, and Baseline+PMM—driven by the eight workload conditions.

Fig. 9 shows the MPR counts of the five schemes handling the eight traces in the simulated SSD. A high number of MPR counts ensures the fast speed of the entire storage system. Fig. 9 indicates that FSPDA achieves the highest parallel improvement under all the tested workloads. For instance, compared with the baseline scheme, our FSPDA pushes up the number of parallel read operations under *DAP*, *CFS*, *24HRS*, *DDR20*, *MSNFS*, *mds_0*, *stg_1*, and *proj_0* by 150%, 11.3%, 20.3%, 48.5%, 39.8%, 2.9%, 6.9%, and 32.8%, respectively. Likewise, compared against OSPADA, SOML, and Baseline+PMM, FSPDA raises the number of MPR operations by an average of 31.6%, 37.2%, and 37.1%, respectively. Although OSPADA also enhances parallelism by scattering continuous data, a major portion of the improved MPR operations are redundant ones. These results demonstrate that FSPDA is superior to the other alternatives in the pile. Unlike the existing schemes, FSPDA distributes consecutive pages to different planes by the cross-allocation method, which effectively alleviates the adverse impact of FSP+MPP operations on the parallel read operation MPRs, thereby reshaping the parallelism of reads in SSDs.

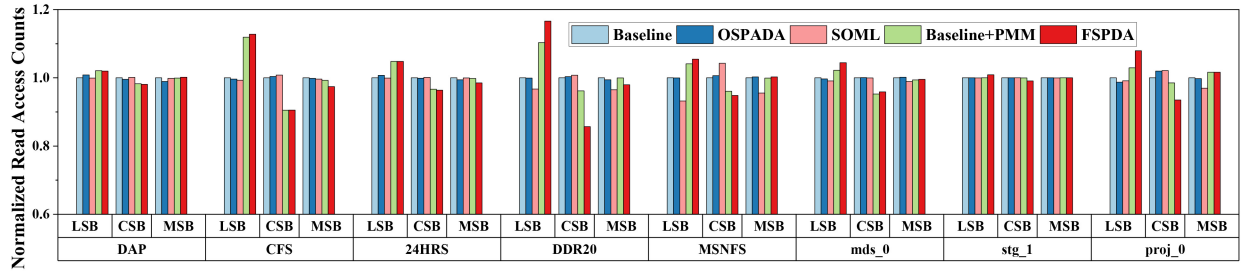


Fig. 10. Read access counts of all the evaluated schemes with eight traces.

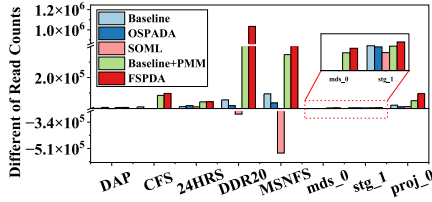


Fig. 11. Difference of read access count with different schemes (LSB-CSB).

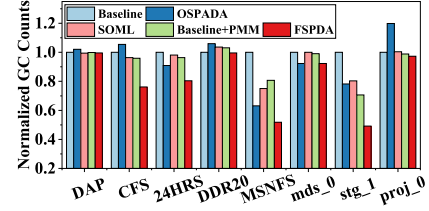


Fig. 12. GC counts of different schemes with eight traces.

2) *Read Access Counts*: Now, we examine the performance of the SSD optimization schemes in terms of read access counts. Fig. 10 unveils the read access counts of the SSD optimization schemes under the eight workloads. We can observe that FSPDA has the highest and lowest read access counts on LSB and CSB pages, respectively. Compared with the baseline scheme, for example, FSPDA drastically enhances the read access counts of LSB pages under *CFS*, *24HRS*, *DDR20*, *MSNFS*, and *proj_0* by 12.7%, 4.8%, 16.6%, 5.4%, and 7.9%, respectively. FSPDA also cuts back the read counts of CSB pages under the five traces by an average of 7.8%. The performance of OSPADA and SOML, being similar to that of the Baseline in terms of read access counts, has fewer read access counts of LSB pages and more CSB pages in some traces. Baseline+PMM only falls short of FSPDA. The core reason is that Baseline+PMM allocates consecutive hot logical pages to a word line, and the page matching module has to place one of the hot pages on the CSB page. Fortunately, FSPDA allocates the pages to different word lines, thus providing more LSB pages to place the pages. Thus, FSPDA effectively improves the read utilization of low-latency LSB pages and suppresses the utilization of high-latency CSB pages.

Moreover, LSB pages have the lowest latency and the highest reliability while CSB has the highest latency and instability; therefore, the read utilization of the LSB and CSB pages immensely influences the read performance. To give an insight into the read access count gap between the two pages, Fig. 11 plots the difference of read access counts between LSB and CSB pages of the five schemes under the eight workloads. A positive number reveals that there are more LSB pages than CSB pages. Conversely, a negative number represents more CSB pages. In the FSPDA case, the read access counts of LSB pages are dramatically higher than that of CSB. Especially, under *DDR20*, FSPDA expends the difference of read access counts between LSB and CSB pages to more than 1000 000, which suggests that FSPDA offers one million more low-latency reads than the other strategies. In short, FSPDA

is capable of significantly shortening read latency by shifting massive read requests from high-latency CSB pages to low-latency LSB pages.

3) *GC Counts*: It is noteworthy that NAND cells can only perform a limited number of erasable operations. However, each garbage collection triggers an erase operation and occupies a large amount of flash resources. Therefore, GC counts are a viable indicator to gauge SSD performance. For this reason, we investigate the impact of FSPDA on SSD lifetime by comparing FSPDA with the other four optimization schemes in terms of GC counts under the eight real-world workloads.

Fig. 12 sketches the GC counts of different schemes under the eight workloads. The findings illustrate that compared with the other four schemes, FSPDA is the most conducive approach to prolonging the SSD lifetime: a low GC count indicates a long SSD lifetime. For instance, FSPDA reduces the number of GC counts of the baseline scheme under traces *CFS*, *24HRS*, *MSNFS*, *mds_0*, *stg_1*, and *proj_0* by 23.8%, 19.6%, 48.1%, 7.6%, 50.8%, and 2.6%, respectively. In the *DDR20* case, while FSPDA exhibits similar performance to Baseline, FSPDA outperforms the other three strategies. FSPDA reduces the GC counts of Baseline+PMM by an average of 12.3%. As expected, the OSPADA scheme has a weak performance in terms of GC counts because the redundant parallelism of OSPADA ignores data locality and thus increases garbage collection overhead. The baseline scheme places consecutive pages of a write request on a word line, thereby improving data locality. Unfortunately, the baseline has a high requirement on the size of a write request. Specifically, when a write request has sufficient consecutive pages (larger write request size), the baseline will have an excellent data locality. Conversely, if the size of the write request is small and the requested pages are heavily fragmented, the data locality provided by the baseline is limited. Fortunately, our FSPDA is not affected by the write request size. The primary reason is that FSPDA determines the physical locations of pages by logical distance rather than the access order of write requests. Therefore, FSPDA places logical pages with close logical distance on adjacent

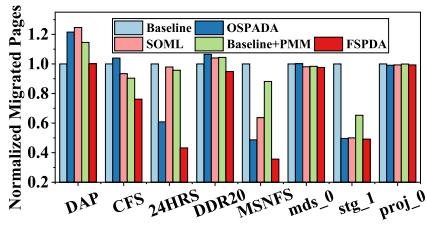


Fig. 13. Migrated pages of different schemes with eight traces.

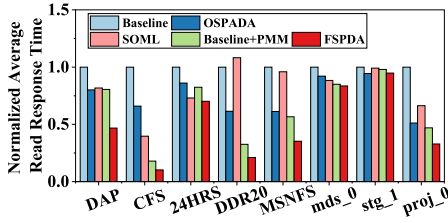


Fig. 14. Average read response time with different schemes.

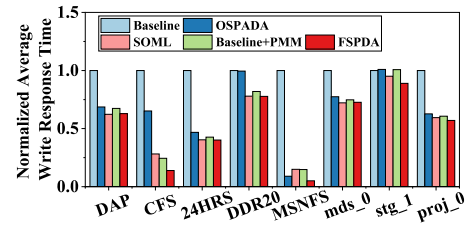


Fig. 15. Average write response time with different schemes.

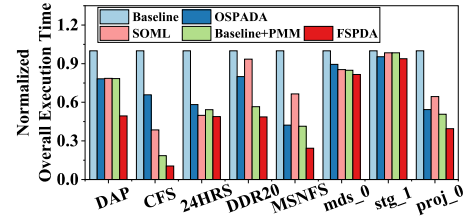


Fig. 16. Overall execution time of different schemes with eight traces.

physical locations regardless of the write request size, which minimizes garbage collection operations by the virtue of high data locality.

4) *Migrated Pages*: Valid page migration—the core operation of garbage collection—has a heavy impact on GC execution time. As such, we compare FSPDA with the other four schemes in terms of migrated valid pages under the eight workloads. The comparison results are plotted in Fig. 13, in which few migrated pages indicate competent GC performance.

We observe from Fig. 13 that FSPDA noticeably lowers migrated valid pages during GC operations. For instance, FSPDA slashes the number of migrated pages under *CFS*, *24HRS*, *DDR20*, *MSNFS*, *mds_0*, and *stg_1* by 23.8%, 56.7%, 5.1%, 64.3%, 2.3%, and 50.7%, respectively. In addition, FSPDA cuts back the migrated pages of OSPADA, SOML, and Baseline+PMM by an average of 11.7%, 16.8%, and 20.1%, respectively. In summary, FSPDA has the least migrated pages, compared with the other schemes, across all the tested workload conditions. This outstanding performance is justified by several compelling reasons. Recall that FSPDA determines the distance of physical locations by the distance of logical pages, and FSPDA enhances read parallelism by the cross-allocation methods. Such an allocation mechanism enables FSPDA to adequately curtail data fragmentation, which is a culprit for massive migrated pages.

5) *Response Time*: Now, we are in a position to assess the response time—an important metric for evaluating storage system performance. We separately gauge average read response time and average write response time, and then evaluate the overall execution time of all completed requests. Again, we conduct experiments to compare FSPDA with the other schemes on the eight traces.

Fig. 14 unravels that FSPDA outperforms the other four competitors with respect to average read response time. Compared with the baseline scheme, FSPDA slashes its read response time on the entire traces by 53.1%, 89.7%, 29.8%, 78.9%, 64.6%, 16.4%, 5.2%, and 67.1%, respectively. Moreover, FSPDA improves the read performance of OSPADA, SOML, and Baseline+PMM by an average of

24.7%, 32.2%, and 13.2%, respectively. Fig. 15 shows that, unsurprisingly, the write performance of FSPDA has a leading edge over the other four schemes regardless of the workload conditions. More prosaically, FSPDA reduces the write response time of the four alternatives Baseline, OSPADA, SOML, and Baseline+PMM by an average of 47.7%, 13.7%, 3.6%, and 6.1%, respectively. In addition, Baseline+PMM also effectively reduces the read and write response time. Especially, in terms of read response time, Baseline+PMM has an obvious advantage over the three scheme Baseline, OSPADA, and SOML. These results are expected as the page-matching module significantly improves the overall latency performance by reducing the read latency of hot read data. However, in addition to the page matching module, FSPDA also integrates the data queuing unit to improve read parallelism and data locality. Thus, FSPDA exhibits the most prominent performance in the metric of response time.

The rationales behind these findings are summarized as follows. First, FSPDA impressively reduces read response time by increasing read parallelism and placing hot data on low-latency pages. Second, albeit OSPADA and SOML optimize read performance, OSPADA is heavily slowed down by garbage collection, whereas SOML fails in speeding up read requests that exceed 16 kB. Third, because write update requests usually spark read operations to maintain data consistency, the low read latency can improve the efficiency of writes updates—reducing write latency [2], [10], [23]. Meanwhile, GCs occupy channel resources for data migration, which will block write requests, thereby significantly increasing write response time. FSPDA allocates logical pages with close logical distance to adjacent physical locations, thereby improving data locality and GC efficiency. The improvement of GC efficiency alleviates the blocking time of write requests. Therefore, FSPDA has the most prominent performance among its peers.

Furthermore, Fig. 16 reveals the overall execution time of different schemes under the eight workloads. The findings indicate that compared with the other four schemes, FSPDA is the most conducive scheme to reduce the overall time overhead

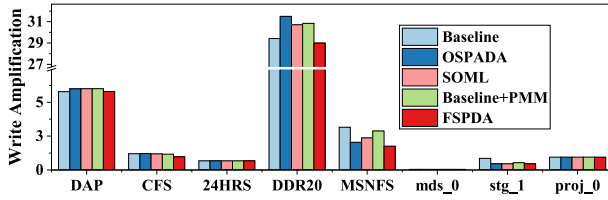


Fig. 17. Write amplification of different schemes.

of processing requests. For example, FSPDA slashes the overall execution time of the baseline scheme under the eight workloads by 50.5%, 89.4%, 51.1%, 51.3%, 75.6%, 18.4%, 6.1%, and 60.5%, respectively. FSPDA reduces the overall execution time of OSPADA, SOML, and Baseline+PMM by an average of 20.9%, 22.3%, and 10.8%, respectively. Meanwhile, Baseline+PMM also has an excellent performance by the page matching module of FSPDA. The experimental results are consistent with our prediction. The reduction of average response time can directly slash the overall execution time. Fortunately, in the average read-and-write response time experiments, FSPDA has the most prominent performance improvement. Therefore, FSPDA can effectively reduce the overall execution time of read and write requests, thereby improving the overall SSD performance.

6) *Write Amplification*: In the last set of experiments, we evaluate the write amplification—one of the key SSD performance metrics—of the four SSD optimization schemes on the eight traces. Generally speaking, due to the impact of data migration and page misalignment, actual write accesses are usually much more than write accesses issued by a host: a small write amplification signifies high SSD performance.

Fig. 17 shows the write amplification measures of the different optimization schemes. The results reveal that FSPDA remarkably lowers the write amplification of the other schemes under *CFS*, *DDR20*, *MSNFS*, and *stg_1* workloads. Under the other traces, FSPDA marginally reduces the write amplification of the other SSD optimization schemes. And OSPADA has the worst write amplification performance under some traces. The reason behind such poor performance is that OSPADA disperses contiguous data and increases data fragmentation causing extra write requests for data migration amid garbage collection. Also, FSPDA reduces the write amplification of Baseline+PMM by an average of 8.7%. These experimental results trumpet that FSPDA furnishes the lowest write amplification among the alternative schemes thanks to the factor that FSPDA is adept at maintaining data locality and curbing migration pages. The reduction of migrated pages excessively alleviates the write amplification issue.

Furthermore, compared with Baseline+PMM, FSPDA implements multiple write-back queues, but its extra space and energy overhead is negligible. As mentioned in Section V-D, the space occupied by the write-back queues is extremely small. Thus, the space overhead of FSPDA over Baseline+PMM is extremely limited. In addition, the time overhead of FSPDA is $O(n)$, where n is the number of queued pages in the write-back queues. Since the size of the write-back queues is extremely small, the time overhead of FSPDA is limited, which has the negligible impact on the runtime and energy overhead of the SSD controller. Meanwhile, the

DRAM energy overhead caused by the write-back queues is also negligible, because DRAM is always used to store data.

In a nutshell, FSPDA is conducive to boosting the performance of multibit 3-D NAND flash-based SSDs. FSPDA optimizes the number of MPR operations of OSPADA and SOML by 31.6% and 37.2%, respectively. FSPDA also slashes the average read response time of OSPADA and SOML by 24.7% and 32.2%, respectively. Furthermore, FSPDA exhibits an outstanding performance in terms of read access counts, GC counts, and page migration.

VII. CONCLUSION

In this study, we proposed an FSP data allocation scheme FSPDA to revamp the overall performance of multibit 3-D NAND flash-based SSDs. FSPDA allocates physical locations for adjacent logical pages according to logical distance, and the cross-allocation method is adopted to optimize the read parallelism of consecutive pages. In addition, FSPDA employs the cache read hits to predict hot data to be placed on low-latency physical pages, thereby shortening read latency of the hot data. By assigning appropriate physical addresses to logical pages, FSPDA not only immensely cuts back read/write latency in flash memory but also curbs the overhead of garbage collection.

We conducted extensive experiments to quantitatively evaluate FSPDA in the context of SSD performance improvement. We compared FSPDA with a baseline dynamic allocation scheme and two state-of-the-art schemes—OSPADA and SOML—driven by eight real-world workloads. The experimental results are encouraging: compared with the existing solutions, FSPDA trims read response time and the number of garbage collections by averages of 13.6% and 28.5%, respectively.

REFERENCES

- [1] W. Liu et al., "Characterization summary of performance, reliability, and threshold voltage distribution of 3D charge-trap NAND flash memory," *ACM Trans. Storage*, vol. 18, no. 2, pp. 1–25, 2022.
- [2] J. Cui, Z. Zeng, J. Huang, W. Yuan, and L. T. Yang, "Improving 3-D NAND SSD read performance by parallelizing read-retry," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 768–780, Mar. 2023.
- [3] S. Nie, Y. Zhang, W. Wu, and J. Yang, "Layer RBER variation aware read performance optimization for 3D flash memories," in *Proc. 57th ACM/IEEE Design Autom. Conf.*, 2020, pp. 1–6.
- [4] F. Wen, M. Qin, P. V. Gratz, and A. L. N. Reddy, "Hardware memory management for future mobile hybrid memory systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3627–3637, Nov. 2020.
- [5] "3D flash memory toggle DDR2.0 technical data sheet," Data Sheet TH58TFxxV23BAxx, TOSHIBA, Tokyo, Japan, 2016.
- [6] 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.
- [7] 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.
- [8] F. Wu et al., "Characterizing 3D charge trap NAND flash: Observations, analyses and applications," in *Proc. 36th IEEE Int. Conf. Comput. Design*, 2018, pp. 381–388.
- [9] J. Lee, M. Kim, W. Choi, S. Lee, and J. Kim, "TailCut: Improving performance and lifetime of SSDs using pattern-aware state encoding," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 409–414.

- [10] C.-Y. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, and C. R. Das, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 955–969.
- [11] I. Fareed, M. Kang, W. Lee, and S. Kim, "Update frequency-directed subpage management for mitigating garbage collection and DRAM overheads," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2467–2480, Dec. 2021.
- [12] S. Wu, B. Mao, Y. Lin, and H. Jiang, "Improving performance for flash-based storage systems through GC-aware cache management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2852–2865, Oct. 2017.
- [13] H. Sun, S. Dai, J. Huang, and X. Qin, "Co-active: A workload-aware collaborative cache management scheme for NVMe SSDs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1437–1451, Jun. 2021.
- [14] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1141–1155, Jun. 2013.
- [15] 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. 25th Int. Conf. Supercomput.*, 2011, pp. 96–107.
- [16] B. R. Reddy, C. Narendra, T. Prakash, V. C. Kavirayani, and P. N. P. Singh, "Method and an apparatus for analyzing data to facilitate data allocation in a storage device," U.S. Patent 9 582 199, Aug. 2017.
- [17] Y. Shim, M. Kim, M. Chun, J. Park, Y. Kim, and J. Kim, "Exploiting process similarity of 3D flash memory for high performance SSDs," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 211–223.
- [18] Y.-M. Chang, Y.-H. Chang, T.-W. Kuo, H.-P. Li, and Y.-C. Li, "A disturb-alleviation scheme for 3D flash memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2013, pp. 421–428.
- [19] Y. Wang, M. Zhang, L. Dong, and X. Yang, "A thermal-aware physical space allocation strategy for 3D flash memory storage systems," in *Proc. Int. Symp. Low Power Electron. Design*, 2016, pp. 290–295.
- [20] N. Shahidi, M. T. Kandemir, M. Arjomand, C. R. Das, M. Jung, and A. Sivasubramaniam, "Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2016, pp. 561–572.
- [21] W. Wang and T. Xie, "PCFTL: A plane-centric flash translation layer utilizing copy-back operations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3420–3432, Dec. 2015.
- [22] C.-Y. Liu, Y. Lee, W. Choi, M. Jung, M. T. Kandemir, and C. Das, "GSSA: A resource allocation scheme customized for 3D NAND SSDs," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 426–439.
- [23] F. Wu, Z. Lu, Y. Zhou, X. He, Z. Tan, and C. Xie, "OSPADA: One-shot programming aware data allocation policy to improve 3D NAND flash read performance," in *Proc. 36th IEEE Int. Conf. Comput. Design*, 2018, pp. 51–58.
- [24] S. Yan et al., "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 15–28.
- [25] A. Tavakkol, P. Mehrvarzy, M. Arjomand, and H. Sarbazi-Azad, "Performance evaluation of dynamic page allocation strategies in SSDs," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 2, pp. 1–33, 2016.
- [26] B. Mao, S. Wu, and L. Duan, "Improving the SSD performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.
- [27] C.-Y. Liu, J. B. Kotra, M. Jung, and M. T. Kandemir, "PEN: Design and evaluation of partial-erase for 3D NAND-based high density SSDs," in *Proc. 16th USENIX Conf. File Storage Technol.*, 2018, pp. 67–82.
- [28] L. M. Grupp, J. D. Davis, and S. Swanson, "The Harey tortoise: Managing heterogeneous write performance in SSDs," in *Proc. 13th USENIX Annu. Tech. Conf.*, 2013, pp. 79–90.
- [29] W. Zhang, Q. Cao, H. Jiang, and J. Yao, "Improving overall performance of TLC SSD by exploiting dissimilarity of flash pages," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 332–346, Feb. 2020.
- [30] R.-A. Cernea et al., "A 34 MB/s MLC write throughput 16 Gb NAND with all bit line architecture on 56 nm technology," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 186–194, Jan. 2009.
- [31] "SNIA IOTTA." MSR Cambridge Traces. 2007. [Online]. Available: <http://iota.snia.org/traces/block-io/388?only=386>
- [32] M. Kwon et al., "TraceTracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction," in *Proc. IEEE Int. Symp. Workload Characterization*, 2017, pp. 87–96.



Shujie Pang received the M.S. degree from the Computer Science Department, Yantai University, Yantai, China, in 2020. He is currently pursuing the Ph.D. degree with the Computer Science Department, Jinan University, Guangzhou, China.

He has published several papers in *ACM Transactions on Storage* and *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*. His current research interests include storage system, computer architecture, and performance evaluation.



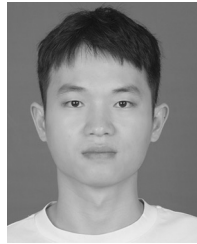
Yuhui Deng received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2004.

He is a Professor with the Computer Science Department, Jinan University, Guangzhou, China. Before joining Jinan University, he worked with EMC Corporation, Hopkinton, MA, USA, as a Senior Research Scientist from 2008 to 2009. He worked as a Research Officer with Cranfield University, Cranfield, U.K., from 2005 to 2008. His research interests cover data management, data analysis, data storage, and cloud computing.



Zhaorui Wu received the M.S. degree in computer architecture from the Computer Science Department, Jinan University, Guangzhou, China, in 2021, where he is currently pursuing the Ph.D. degree.

His current research interests include parallel and distributed computing, computer architecture, and performance evaluation.



Genxiong Zhang received the B.E. degree in computer science and technology from South China Agricultural University, Guangzhou, China, in 2020.

He is a Research Student with the Computer Science Department, Jinan University, Guangzhou. His current research interests include storage system and nonvolatile memory.



Jie Li received the B.E. degree in automation from the Wanjia University of Technology, Ma'an Shan, Anhui, China, in 2016, and the M.S. degree in computer science and technology from the Guangxi University for Nationalities, Nanning, China, in 2019. He is currently pursuing the Ph.D. degree with the Computer Science Department, Jinan University, Guangzhou, China.

His current research interests include parallel and distributed computing, data center architecture, cloud computing, and data replica placement.



Xiao Qin received the B.S. and M.S. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1992 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska-Lincoln, Lincoln, NE, USA, in 2004.

He is currently an Alumni Professor and the Director of Graduate Programs with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA. His research interests include parallel and distributed systems,

storage systems, and fault tolerance.

Dr. Qin won an NSF CAREER Award in 2009.