2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2235-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/HPCA51647.2021.00043

GSSA: A Resource Allocation Scheme Customized for 3D NAND SSDs

Chun-Yi Liu
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
cql5513@cse.psu.edu

Yunju Lee
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
yunju@psu.edu

Wonil Choi
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
wuc138@psu.edu

Myoungsoo Jung
CAMEL, Electrical Engineering
KAIST
Daejeon, South Korea
mj@camelab.org

Mahmut Taylan Kandemir
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
mtk2@cse.psu.edu

Chita Das
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
das@cse.psu.edu

***Abstract*—The high density of 3D NAND-based SSDs comes with longer write latencies due to the increasing program complexity. To address this write performance degradation issue, NAND flash manufacturers implement a 3D NAND-specific full-sequence program (FSP) operation. The FSP can program multiple-bit information into a cell simultaneously with the same latency as the baseline program operation, thereby dramatically boosting the write performance. However, directly adopting the (large granularity) FSP operation in SSD firmware can result in a lifetime degradation problem, where small writes are amplified to large granularities with a significant fraction of empty data. This problem cannot completely be mitigated by the DRAM buffer in the SSDs since the "sync" commands from the host prevent the DRAM buffer from accumulating enough written data. To solve this FSP-induced performance/lifetime dilemma, in this work, we propose and evaluate GSSA (Generalized and Specialized Scramble Allocation), a novel written-data allocation scheme in SSD firmware, which considers both various 3D NAND program operations and the internal 3D NAND flash architecture. By adopting GSSA, SSDs can enjoy the performance benefits brought by the FSP without excessively consuming the lifetime. Our experimental evaluations reveal that GSSA can achieve the throughput and the spent-lifetime of the best-performance and best-lifetime single granularity schemes, respectively.**

***Index Terms*—SSD management, page allocation, 3D NAND flash, scramble allocation**

I. Introduction

Due to the ultra high density and low price per GB, 3D NAND flash [1]–[12] becomes the dominant solid-state storage in the market [13]. 3D NAND flash technology can highly improve the density of the NAND-based SSDs via stacking a number of layers on top of one another – up to 128 layers in some configurations [14]. Additionally, the newly-designed 3D NAND cell can store up to 4 bits information per cell, which is referred to as a quad-level cell (QLC). Such density-improving techniques significantly increase the overall 3D flash density.

However, this high density comes with performance degradation, where the latency to program (write) one bit information into a cell is significantly prolonged. [3], [4] To address this performance issue, the flash manufacturers implement a full-sequence program (FSP) operation, which can program multiple bits *simultaneously* into a cell to improve write throughput. For example, the FSP performed on a QLC cell can gain up to 4 times higher throughput than the baseline program executed on the same cell. Although the FSP itself is not a new idea for NAND flash [15], [16], the new 3D NAND cell [3] has actually made the FSP operation feasible.

The FSP significantly improves the 3D NAND performance, but it can cause a lifetime degradation problem. In particular, current SSD firmware is *not* aware of the FSP operation and can only adopt baseline and multi-plane program (MPP) operation.[1] As a result, to enjoy the full benefits of the FSP, the SSD firmware needs to treat the FSP as the baseline program operation. However, by doing so, the program granularity increases significantly; consequently, *small write requests are all amplified to large program operations*, introducing a serious "lifetime problem". Note that the granularity of the baseline program operation is to write 16KB cells simultaneously with one bit information, whereas the FSP operation on 4-bit QLC cells can write 64KB data in tandem.

To handle this FSP-induced lifetime problem, one can consider two possible solutions: (1) accumulating the small writes in a DRAM buffer until the total amount of written data so far becomes larger than the FSP granularity, and (2) employing some other non-volatile memory (NVM) to process the small writes. As for the former solution, the DRAM buffer can certainly be used to accumulate the small writes. It needs to be emphasized however that, "sync" commands are periodically sent to the SSD to ensure storage data consistency. These "sync" commands force the SSDs to immediately flush the dirty data from *volatile* DRAM into *non-volatile* 3D NAND chips; consequently, the SSDs cannot always accumulate writ-

[1]By adopting superpage [17], the multi-plane program (MPP) can also be viewed as the baseline program granularity.

978-1-6654-2235-2/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00043

ten data up to the FSP granularity. Hence, this approach *cannot* successfully address the mentioned problem.

On the other hand, alternate NVMs, such as 2D NAND and 3DXP [18], can mitigate the small write-induced lifetime problem. However, they are typically much more expensive than 3D NAND. In fact, the least expensive NVM (excluding 3D NAND) one can potentially employ is 2D NAND, which is still 16x more expensive than 3D NAND [13]. For example, replacing 7% of the 3D NAND flash capacity with a 2D NAND flash can double the price of the aggregated NAND chips in the SSD, which makes it a non-viable option for the budget 3D SSDs.

Therefore, *both 3D NAND-based SSD firmware and 3D NAND basic program operations need to be redesigned to utilize different write granularities that are provided by 3D NAND flash*. Motivated by this observation, in this work, we propose GSSA, a novel resource allocation scheme for 3D NAND flash, which considers three program operations (baseline, multi-plane, and FSP) and the specifics of the 3D NAND architecture for programming data.

The main **contributions** of this paper can be summarized as follows:

- We conduct performance/lifetime characterization experiments with synthetic fixed-size random write workloads under different program granularities (baseline, multi-plane, and FSP program), provided by 3D flash. The collected results indicate that the large-granularity programs do *not* perform well on small write requests, and vice versa. Hence, to improve performance without sacrificing lifetime, the SSD firmware needs to consider various program operations.
- Motivated by our characterization results, we propose GSSA, a novel "written-data allocation scheme", which considers all types of program operations, characteristics of the underlying 3D NAND architecture, as well as the program disturbance caused by different program operations. Therefore, GSSA can fully utilize the diverse program operations without excessively consuming the SSD lifetime.
- We evaluate the proposed GSSA allocation scheme against different single granularity allocation schemes and other potential competing schemes. The experimental results collected demonstrate that the throughput of our proposal is 3.7x and 1.04x higher than that of the smallest and largest single granularity allocation schemes, respectively. Furthermore, the spent-lifetime of SSD when using our proposal is only 0.2% higher and 72% lower than the smallest and largest granularity allocation schemes, respectively. These results clearly indicate that GSSA can approach, at the same time, the high performance of the largest granularity allocation scheme and the low spent-lifetime of the smallest granularity allocation scheme.

II. Background

A. SSD Overview

Figure 1 shows the overview of an SSD [19]–[24], that contains three main components: (a) NAND flash chips, (b) a DRAM buffer, and (c) an SSD controller. The NAND chips

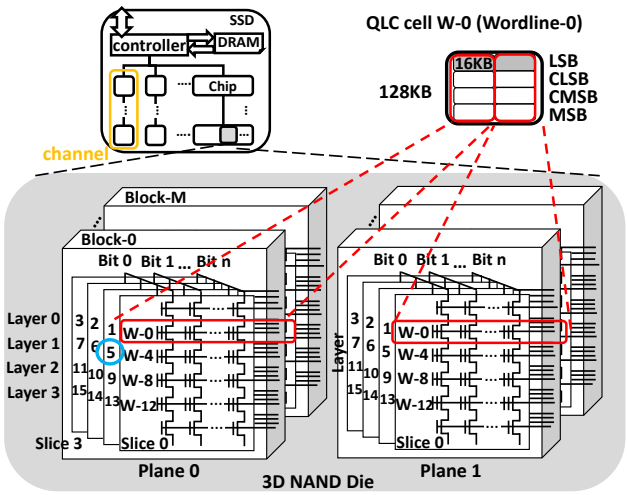
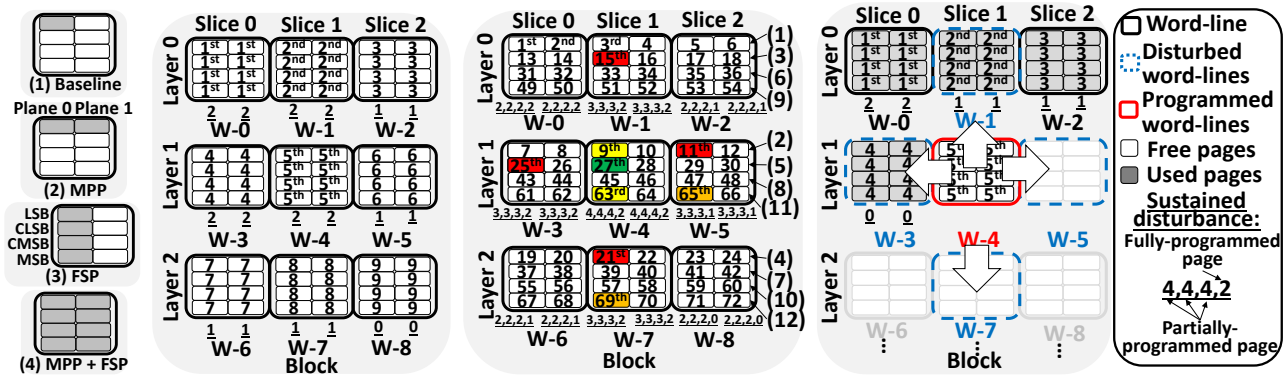


Fig. 1. High-level view of a 3D NAND SSD.

are used for storing data, and a set of NAND chips are serially-connected via a channel. A NAND chip can contain one or more NAND dies.[2] Each NAND die consists of multiple planes (2 in the figure). A plane is composed of thousands of blocks (denoted block-0 to block-M in the figure), and each block contains hundreds of word-lines (denoted W-0 to W-15 in the figure). The word-line organization in the block can vary depending on the NAND type. The figure shows a typical 3D NAND word-line organization, which is a multi-slice and multi-layer architecture. In this organization, to uniquely specify word-line W-5, two indices, layer-index 1 and slice-index 1, are required. A NAND cell in the word-lines can store up to 4-bit information, and in this case, such a cell is referred to as *quad-level cell* (QLC). A QLC word-line can be accessed as 4 separate pages: Least-significant-bit (LSB), Central-less-significant-bit (CLSB), Central-more-significant-bit (CMSB), and Most-significant-bit (MSB) pages (as also shown in Figure 1). Current mainstream page size is 16KB. The NAND flash can only execute one operation at a time, and it provides three basic operations: (a) program (will be introduced shortly), (b) read, and (c) erase operations. The program and read operations are in a unit of page, while an erase operation is performed in a unit of block. Note that the program and erase operations can gradually wear out the NAND cells. When the NAND cells are worn-out, they can no longer store data. Thus, one typically uses the number of program or erase operations to measure the SSD spent-lifetime. In the paper, we use the number of erase operation as the spent-lifetime indicator. More details on NAND flash operations can be found in [19].

The DRAM buffer is mainly used by the flash translation layer (FTL) [25], [26] to store a very large logical-to-physical page mapping table. It can also temporarily store the written data to improve the write throughput. The SSD controller executes the FTL and storage protocols, such as SATA [27] and NVMe [28]. FTL can be roughly divided into two parts: (1)

[2]The NAND chip shown in the figure has only one NAND die.



(a) Program granularities. (b) MPP + FSP allocation. (c) Baseline allocation (Both planes share the same disturbance sequence). (d) Disturbance model.

Fig. 2. The write granularities, the allocation schemes and disturbance model for 3D NAND flash (MPP:Multi-plane program, FSP: Full-sequence program).

logical-to-physical page mapping, and (2) garbage collection (GC). Due to the NAND flash's "erase-before-write" property, FTL employs an "out-place-update" policy to minimize the number of costly erase operations, thereby improving both performance and lifetime. However, this policy can generate a lot of invalid data, preventing the future write operations from being performed. As a result, GC is invoked to clear the invalid data. Further information on GC and related concepts can be found elsewhere [11].

B. Flash Program Operation

To demonstrate the benefits of the FSP, in Figure 2a, we introduce all QLC-based 3D NAND *program (write)* operations, which are (1) baseline program, (2) multi-plane program (MPP), (3) full-sequence program (FSP), and finally (4) combination of MPP and FSP. Note that each of these program operations consists of two stages: (a) transferring to-be-programmed data into the chip internal buffer and (b) programming the data stored in the internal buffer into the NAND cells. The latency range of the first stage (in tens of microseconds) can be ignored compared to that of the second stage (in milliseconds). Hence, our proposal mainly minimizes for the aggregated value of the second-stage latencies.[3]

Let us now discuss the 4 program operations enabled by 3D NAND flash. First, the baseline program operation can only program one page out of the LSB, CLSB, CMSB, and MSB pages in one of the planes. Second, the multi-plane program (MPP) operation can perform the baseline program operation on multiple planes *at the same time*. Third, the FSP can program multiple pages in the same word-line simultaneously. For example, in Figure 2a-(3), the FSP can program the LSB, CLSB, CMSB, and MSB pages of the word-line *simultaneously*. Finally, we can combine MPP with FSP to achieve an 8x throughput over the baseline program operation.

C. 3D NAND Page Allocation

We now introduce two existing 3D NAND (scramble) page allocation schemes: (1) the MPP+FSP and (2) the baseline allocation. Both these schemes adopt a layer-major scheme [29], where the allocated layer index changes only when all of the slice indices have been traversed through once. Figure 2b shows the MPP+FSP (8-page-based) allocation[4], which is good for the large-granular program operations. In the figure, one word-line (i.e., W-4) can be programmed only when all word-lines (i.e., W-0 $\sim$ W-2) in the upper layers and all word-lines (i.e., W-3) left to the word-line in the same layer are programmed. This program order can evenly distribute the program disturbances caused by the program operations. Figure 2d shows the program disturbance model [30]–[32], where a program operation seriously disturbs only the four closest word-lines (upper, left, right, and lower word-lines). Distant word-lines are only slightly disturbed; so, the disturbance caused by these word-lines can be safely ignored.

Specifically, in Figure 2d, the 5^{th} program of this block on W-4 disturbs the four surrounding word-lines, namely, word-lines W-1, W-3, W-5 and W-7. However, under the MPP+FSP allocation scheme, since there will not be any data stored in the right and lower word-lines (W-5 and W-7, respectively) of the programmed word-line, the disturbances on these word-lines can be ignored, and only word-lines W-1 and W-3 are disturbed. Such example shows that a good allocation scheme can significantly reduce the program disturbance.

However, the baseline program operation *cannot* adopt the MPP+FSP allocation scheme since one word-line has to be programmed multiple times (4 times in the case of QLC-based chips) by the baseline program. Such continuous multiple baseline program operations on the same word-line can significantly disturb the surrounding word-lines. For example, as shown in Figure 2d, if 4 baseline program operations were performed back-to-back on word-line W-4, word-lines W-1 and W-3 would suffer from 4 times program disturbance.

Therefore, the NAND flash manufacturers came up with the idea of "scramble allocation" (shown in Figure 2c as our baseline allocation) [29], [33]. The benefit of the scramble allocation is that the later-programmed page in the same word-line can recalibrate the stored values of the previously-

[3]In our evaluations, we modeled both the stages for all program operations to obtain accurate results.

[4]The number shown in each page is the program order in this block.

Algorithm 1: SCRAMBLE LAYER ALLOCATION.

```
layer ← 0;
repeat
    PROGRAM-LAYER(layer);
    for layer_prev ← (layer − 1) to (layer − (bits_per_cell − 1)) do
        if layer_prev >= 0 then PROGRAM-LAYER(layer_prev);
    layer += 1;
    if layer > layer_max then layer ← layer_max ;
until NO-EMPTY-PAGE(layer);
```

programmed pages. As a result, the previously disturbance-induced error bits (caused by other surrounding programs) can be corrected by the later programs. For instance, in Figure 2c, although the data of first page (written by the 9^{th} program) in word-line W-4 are disturbed by the 11^{th}, 15^{th}, 21^{st} and 25^{th} program (write) operations on the surrounding pages, the bit error rate of the first page in word-line W-4 can still be very low, since the later (27^{th}) program operation on the same word-line W-4 recalibrates its disturbed stored values.

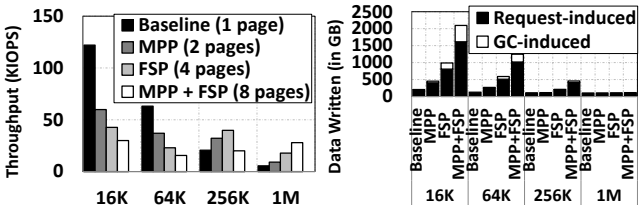
Algorithm 1 shows the algorithm used to decide the scramble allocated layers/pages. Note that, as mentioned earlier, the scramble allocation is applied in a layer-major order, where the layer-allocated order of Figure 2c is shown at the right-most indices. The basic idea of the layer-based scramble allocation is to choose the lowest empty layer (with a highest layer index) to program the data, and then it programs the higher $(bits\_per\_cell - 1)$ layers. The algorithm stops when there is no empty layer/page that can be programmed.

Figure 2c shows the scramble layer/page allocation order and the corresponding number of program disturbances from the surrounding program operation. The value sequences with underlines indicate the number of surrounding program operations (with their induced disturbances). For example, word-line W-4, which is one of the severely-disturbed word-lines, shows that the LSB (written by 9^{th} program) and MSB (written by 63^{rd} program) pages are, respectively, disturbed by 4 (11^{th}, 15^{th}, 21^{st}, and 25^{th}) and 2 (65^{th} and 69^{th}) surrounding program operations. Such allocation scheme involve allocation constraints where partially- and fully-programmed word-lines are designed to tolerate 4 and 2 surrounding programs, respectively. As a result, other allocation schemes satisfy such constraint can ensure that the stored data has at least the same degree of reliability. For instance, as mentioned earlier, the MPP+FSP allocation ensures that a fully-programmed word-line can be at most disturbed by 2 (right and lower) programs; hence, such allocation scheme is reliable.

In summary, two state-of-the-art allocation schemes favor different granular program operations; but, the real-world workloads impose a diverse set of granularity write requests. Hence, to carefully utilize different program operations, a better disturbance-aware allocation scheme needs to be explored.

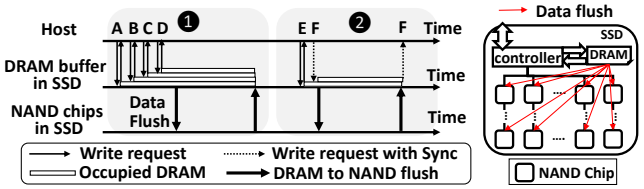
III. MOTIVATION

As mentioned in Section II-B, the full-sequence program (FSP) operation can highly improve the 3D flash write throughput; however, using such large granularity can significantly increase the amount of written data, thereby reducing the SSD lifetime. To illustrate such dilemma, we quantitatively



(a) Performance. (b) Written to NAND chips.

Fig. 3. Comparisons across 4 different write granularity baseline allocation schemes.



(a) Two scenarios. (b) Block diagram.

Fig. 4. Flushing the written data from the DRAM buffer to the NAND chips.

compare the performance and spent-lifetime under different synthetic workloads in Figures 3a and 3b, respectively. These comparisons are conducted by executing random fixed-size write workloads on 8 3D NAND flash directly with four different fixed-size program granularities (shown in Figure 2a).[5] The detailed parameters of the 3D NAND flash and its organization can be found in Table I in Section VI-A.

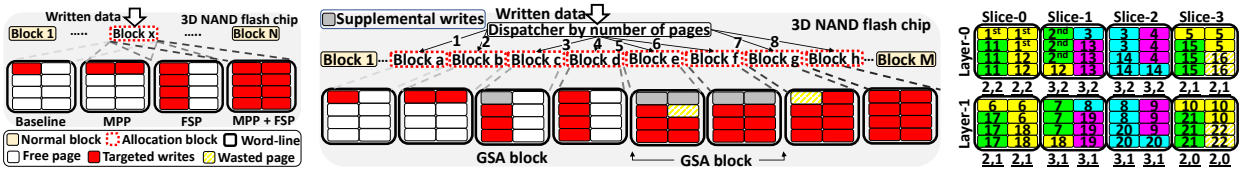
As shown in Figure 3a, under large fixed-size workloads (1M), the throughput becomes higher as the write granularity increases, and it reaches its maximum value with the MPP + FSP program granularity. These results indicate that a large program size can write more data (at a time) into the flash; and, as a result, a higher throughput can be achieved. In contrast, under small fixed-size workloads (16K), the throughput of a smaller write size outperforms that of a larger write size. This is due to the excessive number of garbage collections (GC) caused by a large program size (to be explained shortly). Specifically, owing to the use of a large write size (MPP + FSP), small write requests are written into flash with a large amount of *empty data*. As a result, under the same amount of host-written data, more data are written with a large program size, thereby triggering more GCs.

Figure 3b plots the total amount of data being written into the flash, which includes both the request-induced data and the GC-induced data.[6] Note that all the workloads tested write the same amount of data, which is 100GB. The amounts of actual written data under the large granularity workloads (1M) are nearly the same across four different write sizes. In contrast, the MPP + FSP granularity suffers from a large amount of written data (about 20x of the request data), which is mainly contributed by the large amount of empty data in the MPP + FSP granularity writes.

One may think that a large DRAM buffer that exists in the

[5]Note that the DRAM buffer is *not* considered in Figure 2a to clearly demonstrate the 3D flash performance/lifetime dilemma.

[6]Note that a GC operation needs to copy valid data from the victim block before erasing the block. These copy operations introduce additional written-data.



(a) Overview of the baseline allocation.

(b) Overview of the generalized scramble allocation on 2-plane 4-slice QLC 3D block.

(c) 3-page SSA on 2-plane 4-slice QLC 3D block.

Fig. 5.

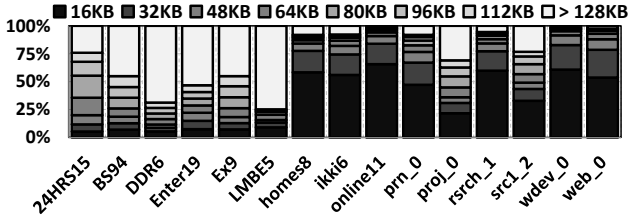


Fig. 6. The write size distribution for the "small" write-dominated workloads.

state-of-the-art SSDs can successfully address the mentioned problem. In reality, a large DRAM can only partially address the problem due to the "sync" commands issued with the writes. To explain the reasons, let us discuss the two examples shown in Figure 4a. ❶ shows a case where the DRAM buffer can successfully accumulate small writes (request A-D). And later, those written data are programmed into 3D flash via the large granularity program operations in the background. Hence, the latencies of those writes can be as short as DRAM access latencies (as opposed to long flash program latencies).

However, the SSD cannot always postpone the flash program operations, even with an unlimited DRAM capacity. This is because the hosts can issue writes with "sync" commands[7] to *force* the SSD program the written data directly to flash chips. Note that file-systems and databases issue such "sync" commands to ensure data consistency. In the case of ❷, the SSD receives a write request (request F) with a "sync" command. In this case, the SSD has to guarantee that the data of not only request F but also all previous requests are written into the flash chip, before returning "finished" to the host. As a result, request F has to wait for the completion of those flash program operations. Even if the host typically issues the write with the "sync" command once a few seconds, the performance degradation can still be observed due to the long-latency program operations. A detailed quantitative analysis can be found in Section VI-B4.

Beside the performance, the "sync" commands can also degrade the SSD lifetime. Figure 4b shows the high-level block diagram of the tested SSD. In the figure, the data in the DRAM is flushed to the flash chips based on the channel-chip-die flash write policy [35].[8] Figure 6 plots the size distribution of writes flushed from the DRAM buffer to flash chips in the SSD (shown in Figure 4b). These 16 represented storage workloads are selected from OpenStor [38], and their detailed properties can be found in Table II in Section VI-A. As can be seen, the large DRAM-equipped SSD still issues a large number of small writes to flash chips under the last 9 workloads. Note that these small writes *cannot* be programmed by large-granular operations since doing so can cause SSD lifetime degradation.

Therefore, to mitigate the performance/lifetime dilemma, in this work, we propose a new "page allocation scheme", which fully utilizes the diverse write granularities offered by 3D flash, to improve performance without excessively sacrificing lifetime.

IV. Overview of GSSA

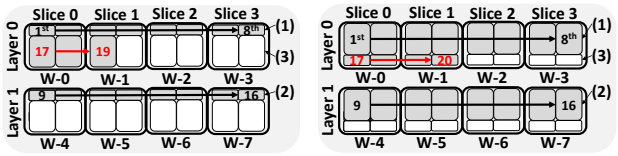
As discussed in Section III, the four uni-granularity allocation schemes (shown in Figure 5a) can fall short for certain types of workloads. Therefore, to achieve the full benefits from the different program granularities brought by state-of-the-art 3D NANDs, we propose a novel Generalized and Specialized Scramble Allocation scheme, *GSSA*.

The core concept behind GSSA is to service the write requests with the minimum number of programs without extensively consuming the 3D flash lifetime. Specifically, GSSA adopts the generalized scramble allocation (GSA) and the specialized scramble allocation (SSA) blocks for all possible program granularities. As a result, most writes can be finished in one operation without programming empty data. Let us begin with the GSA. Figure 5b shows only our GSA proposal, where we prepare n blocks for each n possible program granularities, so that the different granularity writes can be serviced by different allocation blocks. The figure shows our GSA mechanism on 2-plane QLC-based 3D flash, which can have 8 (2 (planes) * 4 (bits per cell)) types of program granularities and needs 8 allocation blocks (shown as block-a $\sim$ block-h). Since 1-, 2-, 4- and 8-page program granularities can easily be achieved by the existing single granularity programs, our discussion mainly focuses on 3-, 5-, 6-, and 7-page program granularities.

To handle those non-trivial program granularities (3-, 5-, 6- and 7-page), we propose that an allocation block can accommodate more than one type of program granularity. For example, as shown in Figure 7a, the data of a 3-page GSA block can be programmed at the units of both 1 page and 3 pages. Hence, as long as the LSB pages of the word-lines are programmed by 1-page writes, the three remaining pages can be programmed by the FSP in one-shot. We call the 1-page writes "supplemental writes", which are used to enable the remaining 3-page writes. The same GSA technique can be easily applied to 5- and 6-page GSA blocks as well.

[7]For example, in SCSI standard [34], a "sync" command can be issued by a write with the FUA flag on.

[8]Such static data allocation scheme can ensure a better sequential read performance since the written data are evenly distributed across differ flash chips based on the logical address. In comparison, dynamic data allocation [36], [37] cannot ensure such property.



(a) Type 1. (b) Type 2.

Fig. 7. 3-page GSA on 2-plane 4-slice QLC 3D block.

However, GSA cannot ensure that no empty data is programmed due to (1) the symmetric MPP operations and (2) insufficient supplemental writes required by the GSA blocks. That is, a majority of flash chips only provide symmetric MPPs, where all planes have to program the same relative position (page index) simultaneously. Hence, we cannot program 2 pages on one plane and 3 pages on another one simultaneously. As a result, we always waste one empty page while performing 5-page programming (block-e in Figure 5b). Regarding the insufficient supplemental writes, the targeted and supplemental writes have to be programmed in order to minimize the program disturbance. As a result, under such extreme uni-granularity (i.e., 3-page write only) situation, the GSA blocks can be useless. This is because the GSA blocks always wait for the supplemental writes.

To address these two sub-optimal scenarios, we propose to release the MPP constraint on 3D NAND and propose specialized scramble allocation (SSA) blocks. Figures 5c and 8 show, respectively, the one possible page program sequence of 3-page and 6-page SSA blocks on a 2-plane 4-slice 3D QLC blocks. As can be seen, with the support of the asymmetric MPPs and the new program sequences, the number of supplemental writes can be minimized under 3-page and 6-page writes. Such SSA blocks can significantly reduce the potential wasted empty data programming. Overall, Figure 9 shows GSSA (consisting of both GSA and SSA) for a 2-plane 4-slice 3D QLC block.

Although the SSA blocks are good for both performance and lifetime, not all write granularities, such as the 7-page write in Figure 9, can find a corresponding SSA block. As a result, we need to combine the SSA and GSA schemes into the *GSSA scheme*. Note that finding a feasible SSA allocation is not trivial due to the program disturbance constraint (mentioned in Section II-C); so, we provide 3 strategies to find the constraint-satisfied SSA blocks in Section V-B1. Also, we discuss the overheads brought by our GSSA proposal in Section V-D.

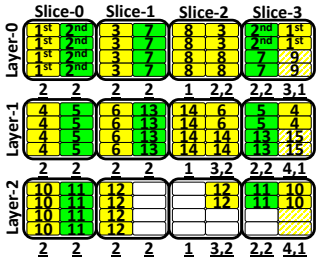


Fig. 8. Specialized 6-page scramble allocation on two-plane 4-slice QLC 3D block.

V. Details of GSSA

A. Generalized Scramble Allocation (GSA)

As mentioned in Section IV, GSA can improve the write performance without sacrificing too much lifetime, but GSA has two design problems: (1) the order of the supplemental writes and the targeted writes and (2) mismatched number of two types of granularities. As shown in Figure 5b, two categories of 3-page GSA blocks can be implemented; however, they result in different write performances. To explain that, we have to introduce the latency of (FSP) program operations. The (full-)FSP is known to have the same latency as the baseline program operation [3]. However, the partial-FSP, which is performed on the partially-programmed word-line, needs a longer overall latency. This is because the (partial-)FSP has to read all previously-programmed data on the word-line to calculate the programmed voltages for the NAND cells. Hence, the overall latency of FSP is calculated as follow:

$$\text{FSP latency} = \text{program latency} + \sum^{previous\ pages} \text{read latency}.$$

From this equation, we know that, to have shorter FSP latencies, the to-be-programmed word-lines should have as few number of programmed pages as possible, thereby introducing a shorter read overhead. As a result, the type 1 of 3-page GSA block (shown in Figure 7a) is better than the other one (shown in Figure 7b), where the word-lines always have only one programmed pages.

Regarding the mismatched numbers of "two types of granularities", one may encounter a situation where the number of one type of writes is many more than that of another one, which can cause sub-optimal resource utilization. That is, the GSA block can wait forever for the supplemental writes to perform the future targeted writes; as a result, these writes can be appended empty data and be dispatched to other large-granularity blocks. One may think we can keep these word-line partially-programmed so that the GSA can continue accommodating the future targeted writes. However, such approach can introduce reliability issues for the stored data [39], which can potentially have a short retention time. Therefore, we propose the SSA blocks, which can minimize the impact of this problem.

Though GSA blocks have these two potential drawbacks, we still need GSA blocks for some non-trivial granularities. Algorithm 2 shows the pseudocode to generate a sequence of program orders to program a GSA block. As mentioned earlier in Figure 7a, every word-line in the GSA block accommodate in order the two types of writes: (1) supplemental write and (2) targeted write. As a result, the program order of a GSA block can be generated by writing the supplemental writes to every word-line of l layer before writing the targeted writes to every word-line of $(l-1)$ layer. The detailed algorithm with two configurable parameters (number of slices (S) and layers (L)) can be found in Algorithm 2, which also consider the boundary conditions (the first and the last layers of wordlines).

Algorithm 2: Generalized Scramble Allocation Program Order Generation.

Input: S: number of slices, L: number of layers
1 **for** $slice \leftarrow 0$ *to* $S-1$ **do** SUPPLEMENTAL-WRITE-ORDER(0, $slice$) ;
2 **for** $layer \leftarrow 1$ *to* $L-1$ **do**
3 **for** $s \leftarrow 0$ *to* $S-1$ **do** SUPPLEMENTAL-WRITE-ORDER($layer$, s) ;
4 **for** $s \leftarrow 0$ *to* $S-1$ **do** TARGETED-WRITE-ORDER($layer-1$, s) ;
5 **for** $slice \leftarrow 0$ *to* $S-1$ **do** TARGETED-WRITE-ORDER($L-1$, $slice$) ;

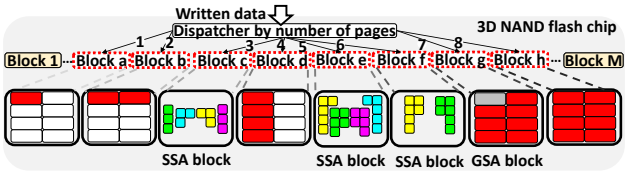


Fig. 9. Overview of the specialized scramble allocation (SSA) on 2-plane 4-slice QLC 3D block.

The example of generated program order of 3-page and 7-page GSA blocks can be found in Figures 7a and 11, respectively.

B. Specialized Scramble Allocation (SSA)

The basic idea behind the SSA scheme is to adopt asymmetric MPP and FSP simultaneously to program the non-trivial granularity writes, such as 3, 5, and 6 pages on a two-plane 3D QLC NAND flash, in one-shot. For example, in Figure 9, 3-page (block-c) writes can be fulfilled by the following four asymmetric MPPs on two planes [plane 0, plane 1]: (1) [3 pages, none] (green), (2) [2 pages, 1 page] (blue), (3) [1 page, 2 pages] (yellow) and (4) [none, 3 pages] (purple). And then, it can easily come up with an allocation (shown in Figure 5c with the same colors), where the four type of MPPs are repeatedly used to service 3-page writes. Also, such an allocation scheme completely satisfies the program disturbance constraint (mentioned in Section II-C): $\underline{4,2} > \underline{3,2}$ (the worst-case sustained-disturbance is on slice-1 and slice-2 of layer-0).

Actually, not all SSA blocks are as easy as the 3-page SSA block. For instance, a constraint-satisfied 5-page SSA block (shown in Figure 10-6) cannot be easily generated. This is because the 5-page granularity is larger than the 4-page QLC word-line capacity, and as a result, some of the word-lines in 5-page SSA block are only programmed once, such as all word-lines on slice-1 in Figure 10. That is, word-line in a block can be programmed once or twice depending on the allocations, which in turn makes finding a feasible allocation scheme very tricky. Therefore, we propose some general strategies to find the feasible allocation scheme.

1) Strategies to Find the SSA Allocation: As mentioned in Section IV, it is not easy to find the SSA block for some non-trivial program granularities, such as 5-page and 6-page writes. This is because of (1) the program disturbance across word-line within a 3D NAND block (mentioned in Section II-C) and (2) the fact that not all word-lines in a block are programmed the same number of times. To solve the two challenges, we propose the following three strategies.

(1) Same allocation order for every layer: A state-of-the-art 3D NAND can have up to hundred of layers of NAND cells; as a result, different allocation orders across different layers can increase difficulty in program disturbance constraint-validation. Hence, we propose to ensure that each and every layer has the same allocation, which is shown in Figures 10 and 8.

(2) Double-end allocation in the same layer: Due to the high complexity of constructing the cross-layer signals to index the word-lines across different layers, the 3D NAND

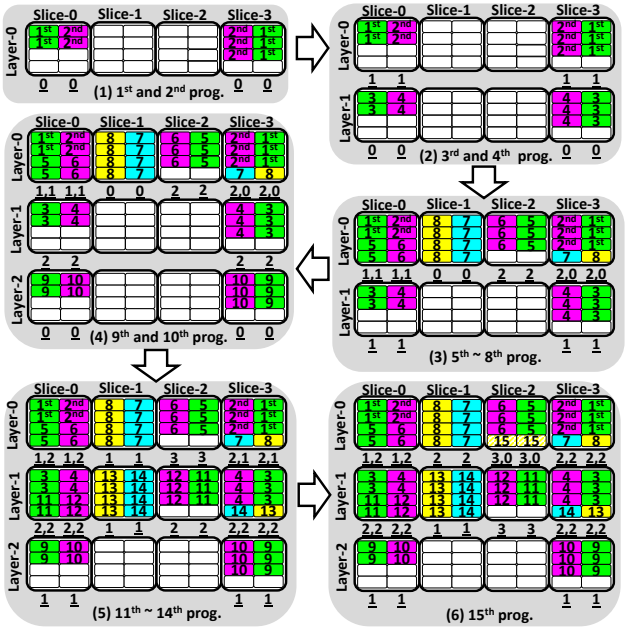


Fig. 10. The 5-page specialized scramble allocation (SSA) on a 4-slice QLC 3D block in 2-plane die.

manufacturers inevitably have to share these per-layer cross-layer selectors across multiple word-lines. However, such multi-word-lines per layer architecture is not fully utilized by the state-of-the-art allocation algorithms[9]. In fact, most of allocation algorithms only allocate the word-lines in the same layer in a monotonic direction (i.e. from the left-most to the right-most word-lines). Actually, performing out-of-order allocations in a layer with the asymmetric MPP and FSP operations under such architecture can achieve good performance without sacrificing the lifetime.

Among various potential 3D asymmetric MPP operation implementations[10], we only require the same-layer asymmetric MPP operations, where the asymmetric MPP operations need always to be on the same layer in the same block. Such design is due to the high-complexity of the cross-layer signals. Later, in Section VI-B, we show that the same-layer asymmetric MPP operations are sufficient to deliver good performance and lifetime. Therefore, with out-of-order allocation in a layer and the same-layer asymmetric MPP operations, we can then adopt the double-end allocation sequence to mitigate disturbance to achieve the desired performance and spent-lifetime.

(3) Switching to another layer to balance the program disturbance: To ensure a (partially-/fully-programmed) word-line only sustain their sustainable disturbance ($\underline{4,2}$ time of neighbor programs), the program order across different layers have to be considered. Note that the MPP can only be programmed in the same layer. Thus, we adopt the layer-changing

[9]Shibata et al. [8] and Shim et al. [40] utilize the similarity of NAND cell physics of the same-layer word-lines to reduce the program latencies, thereby improving performance.

[10]Asymmetric MPP operation is not a new idea for 2D NAND, since some NAND manufacturers already implement such operation and various approaches [41] have been proposed to exploit such operation.

Algorithm 3: GSSA FOR 2-PLANE 3D QLC FLASH

```
Input: Data: written data
switch Data.GET-NUMBER-OF-PAGES() do
    case 1 do
        if CHECK-WRITE-BLOCK-G (Data) then break;
        WRITE-BLOCK-A (Data); break;
    case 2 do WRITE-BLOCK-B (Data); break;
    case 3 do if CHECK-WRITE-BLOCK-C (Data) then break;
    case 4 do WRITE-BLOCK-D (Data); break;
    case 5 do if CHECK-WRITE-BLOCK-E (Data) then break;
    case 6 do if CHECK-WRITE-BLOCK-F (Data) then break;
    case 7 do if CHECK-WRITE-BLOCK-G (Data) then break;
    case 8 do WRITE-BLOCK-H (Data); break;
```

allocation technique of the baseline scramble allocation, where the programs are gradually performed on each layer of word-lines to evenly distribute the program disturbance across all word-lines in a block.

SSA Example: To clearly show the effectiveness of our proposed three strategies on designing a constraint-satisfied allocation, we provide a comprehensive example in Figure 10, which is an allocation order for the 5-page program (block-e in Figure 9). First, as shown in Figure 10-6, the final allocation schemes for each layer are the same. Therefore, as long as we validate the program-disturbance constraints of the first few layers allocation, we can be sure that the remaining layers can satisfy the same constraints.

Second, as shown in Figure 10-1, our allocation scheme allocates pages from the two ends (slice-0 and slice-3) and toward to the middle. The benefit of such approach is that it reduces the program disturbances to and from other word-lines. Third, as plotted in Figure 10-2, instead of allocating the pages from layer-0, our allocation scheme directly allocates the pages from layer-1. This is because the later allocations on layer-0 can completely program some word-lines, such as both word-lines in slice-0 and slice-1; consequently, these word-lines cannot sustain too much disturbance from other neighbor programs. To mitigate that, our allocation adopts the third strategy, which programs the written to the other layers. A similar situation is observed in Figure 10-4 as well.

Allocation example: To demonstrate the core idea of GSSA, Algorithm 3 and Figure 11, respectively, show the GSSA algorithm and the corresponding allocation example on 2-plane 3D QLC NAND die. In the algorithm, the write requests are dispatched to the allocation blocks according to the number of written pages (combining multiple write requests) at a time. Note that the non-trivial write granularities (3-, 5-, 6-, and 7-page writes) may not be able to program on the specific

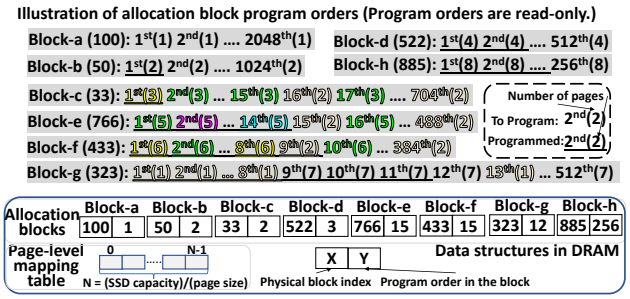


Fig. 11. The allocation example on a 2-plane 3D QLC NAND die.

allocation blocks due to insufficient supplemental writes.

As shown in Figure 11, the program sequence of all blocks are generated at design time, and the program order of each block has to be strictly followed to satisfy the mentioned allocation constraint. Due to the strict program orders, at any given time, each allocation block can only be programmed at one location with the corresponding granularities. Specifically, in the example, we can only program (1) 1^{st} prog of block-a, (2) 2^{nd} prog of block-b, (3) 2^{nd} prog of block-c (Figure 5c), (4) 3^{rd} prog of block-d, (5) 15^{th} prog of block-e (Figure 10), (6) 10^{th} prog of block-f (Figure 8), (7) 12^{th} prog of block-g, or (8) 256^{th} prog of block-h for the next write. As can be noted, we cannot perform a 5-page program on block-e (5-page SSA block) immediately, since the 15^{th} program of block-e is actually a (2-page) supplemental write. As a result, according to Algorithm 3, such write has to dispatched to block-f instead. Further write related discussion can be found in Section V-C.

Allocation block metadata: To track the statuses of allocation blocks, GSSA only has to record two information for each allocation blocks per die in DRAM buffer: (1) physical block index and (2) next available program order. The bottom of Figure 11 shows the mentioned data structure. With such compact in-DRAM data structure per allocation block, GSSA can immediately decide the targeted pages to perform the program operation without reading any additional information from storage. However, One may concern the volatility of DRAM buffer, where the data can get lost due to the SSD power-off. Actually, these metadata can be restructured via SSD startup block scanning. Specifically, during SSD reboot, the SSD controller can read the block-related information stored in its page spare area to infer the allocation block metadata. Note that such method only needs that every block records an additional information, the type of allocation blocks (one of blocks-abcdefgh in our allocation example). Note also that GSSA still needs to keep a copy of the program orders (shown on the top of Figure 11) for each allocation block owing to quick lookups. Actually, these program orders are read-only data and shared by all dies; so, the storage and DRAM footprint (tens of KB) overheads are negligible.

In summary, with the proposed strategies, we can easily find a proper allocation for the most of SSA blocks. Additionally, Figure 12 shows that the proposed strategies can also be applied to the 2-plane 4-slice PLC (Penta-level cell) 3D block, which are known to be a potential 3D NAND design due to the increased high-density demands. This figure reveals that our proposal can be generally applied to various types of 3D NAND flash.

C. Discussion

Wear-leveling (WL): GSSA can combine with any state-of-the-art WL algorithm designed for the page-level mapping [42]–[45]. This is because GSSA only changes the number of allocation blocks per die and the allocation order in each allocation block. Hence, the rest of non-allocation blocks can naturally be managed by these WL algorithms.

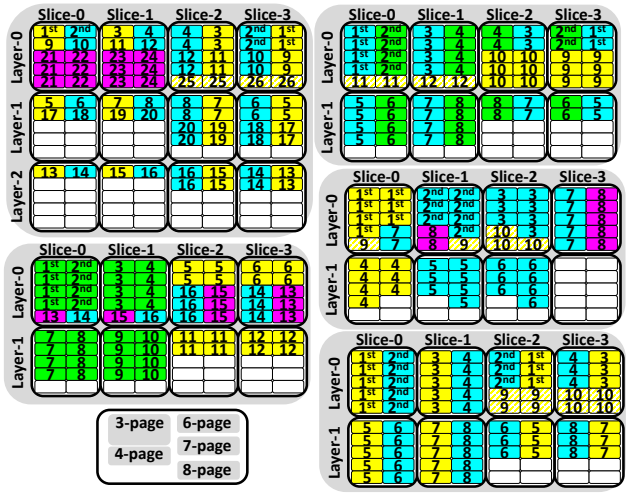


Fig. 12. The 3-/4-/6-/7-/8-page specialized scramble allocation (SSA) blocks on a 4-slice PLC (Penta-level cell) 3D block in 2-plane die. Note that 1-/2-/5-/10-page writes are the trivial cases, while 9-page write can only be allocated by the corresponding generalized scramble allocation (GSA) block.

Supplemental writes: Both GSA and SSA blocks need the supplemental writes to properly allocate pages for different granular writes. Since the amount of supplemental writes need for GSA and SSA blocks are different, they adopt different strategies. For SSA blocks, which needs fewer supplemental writes, GSSA (1) writes empty data for the SSA supplemental writes in the background or (2) writes the data to the allocation blocks designed for the larger targeted writes. The example is shown in Figure 11. On the other hand, for GSA blocks, which demand more supplemental writes, GSSA waits for the actual writes for them. Such strategy can be observed in Algorithm 3, where 1-page write always tries to write to 7-page GSA block (block-h) first, and then block-a.

D. Overhead Analysis

The proposed GSSA has two major sources of overhead: (1) additional allocation blocks and (2) allocation block selection overhead. GSSA requires at least $B * P$ (4 * 2 = 8 for a 2-plane QLC 3D flash die) additional allocation blocks for each flash chip, which only slightly increases the SSD management overheads. One may also be concerned that those additional allocation blocks can increase the difficulty of maintaining crash consistency. We believe that this difficulty is still manageable since some state-of-the-art FTLs, such as Superblock FTL [26], adopt many more allocation blocks. Consequently, GSSA can adopt such techniques to ensure crash consistency. Regarding the allocation block selection overhead, we show the quantitative comparison in Section VI-B.

VI. Evaluation

A. Experimental Setup

We augmented the SSDsim [35] simulator to model the multi-layer and multi-slice architecture of a 3D NAND flash. Following the trend in industry, the evaluated SSD has 8 3D NAND chips (note that each chip has a very high density),

64-layer 4-slice 3D QLC NAND chip parameters [3]	
(Die, Plane, Block, Page, Page size)	(1, 2, 4096, 1024, 16K)
(Program latency, Erase latency)	(1.3ms, 10ms)
Read latency (LSB-...-MSB)	(90, 120, 150, 180)μs
Program induced read overhead	20μs
Chip capacity	128 GB
SSD parameters	
(channel, chips, DRAM capacity)	(4, 8, 64MB)
(FTL, GC trigger)	(Page-level, 5%)
Victim block selection	Max #invalid pages
Transfer time per byte	5ns
(Over provision, Initial data)	(5%, 95%)
DRAM replacement policy	LRU replacement

TABLE I
CHARACTERISTICS OF THE EVALUATED SSDS.

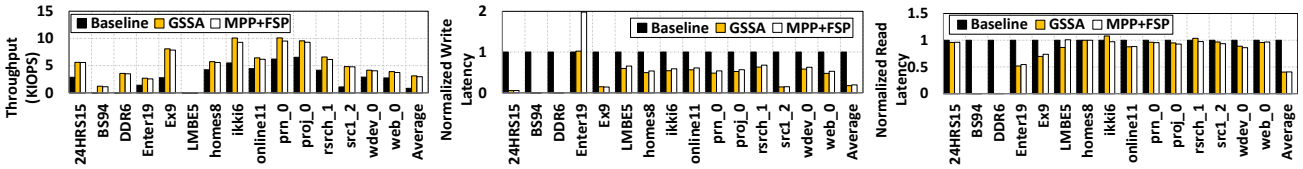
Trace	W	C	A	S	R	Trace	W	C	A	S	R
24HRS15	31	3.8	109	0	152	**online11**	77	0.5	27	3.9	354
BS94	32	4.3	96	0	114	**prn_0**	77	47	48	2.4	10
DDR6	93	22.8	108	0	72	**proj_0**	94	147	81	2.5	14
Enter19	14	20.7	95	0	2	**rsrch_1**	100	0.16	32	6.8	4224
Ex9	78	7.1	97	0	108	**src1_2**	83	45	71	4.7	32
LMBE5	17	6.1	50	0.04	68	**wdev_0**	72	7.3	29	6.8	52
homes8	41	0.8	36	2.7	246	**web_0**	40	11	34	5.3	30
ikki6	100	0.15	33	3.9	1604						

TABLE II
IMPORTANT CHARACTERISTICS OF OUR WORKLOADS (W: WRITE RATIO(%), C: WRITE COVERAGE (GB), A: AVERAGE WRITE SIZE (KB), S: SYNC RATIO IN WRITES(%), R: TRACE REPLAY TIMES.

and consequently, we only need 8 128GB chips to construct a 1 TB SSD. Other relevant SSD and 3D NAND flash related parameters are listed in Table I. The evaluated storage workloads are downloaded from OpenStor repository [38], and include 600+ storage workloads. We want to emphasize that, our evaluations tested *all* OpenStor workloads; however, due to limited space, we only present the results for a selected set of representative workloads (shown in Table II).

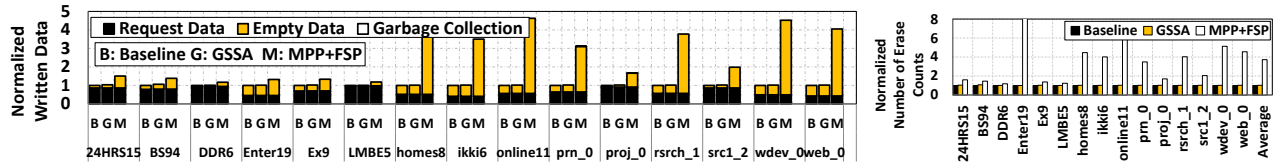
B. Evaluation Results

1) Comparison against the Baseline Schemes: This comparison shows that GSSA can achieve the same high performance as the MPP + FSP allocation, and at the same time the same low SSD lifetime consumption as the baseline allocation.
Performance: Figure 13a gives the throughput comparison of our proposal, the baseline allocation, and the MPP + FSP allocation. As can be observed from this plot, GSSA can achieve a slightly better performance than the largest granularity (MPP + FSP) allocation. This is because GSSA reduces the amount of written data; consequently, fewer number of garbage collection is performed, resulting in a better performance. On average, GSSA outperforms the baseline and MPP + FSP allocations by 3.7x and 1.04x, respectively.
Write/Read Latency: Figures 13b and 13c plot the average write and read latency values, respectively, when using GSSA, the baseline, and the MPP + FSP allocation. In general, the results in Figure 13b indicate that the average write latency of GSSA is as good as that of the MPP + FSP allocation. The only outlier is workload Enter19, where the average write latency of the MPP + FSP allocation is much higher than that of our GSSA. The reason is that, this is a very large workload, which induces a large number of garbage collection (GC) operations under all allocation schemes tested. The largest granularity (MPP + FSP) allocation amplifies the written data, and as



(a) Throughput. (b) Write latency. (c) Read latency.

Fig. 13. Performance comparison across the baseline, GSSA and the MPP + FSP allocation schemes.



(a) Amount of written-data. (b) Number of erase counts (spend-lifetime).

Fig. 14. Lifetime comparison across the baseline, GSSA, MPP + FSP allocation schemes.

a result, it triggers many more GCs, thereby significantly increasing the write latency.

Figure 13c shows that, under our scheme, the average read latency is also highly reduced compared to the baseline allocation on some workloads, such as BS94 and DDR6. This is because, under the baseline scheme, the number of program operations is much higher than the other two schemes. As a result, more read requests are blocked by the program operations, prolonging their read latencies.

Amount of Written Data and Number of Erase Operations (Spent-Lifetime): Figures 14a and 14b plot, respectively, the normalized amount of written data and the number of erase operations issued to the NAND chips under GSSA, the baseline and the MPP + FSP allocation schemes. As can be observed from Figure 14a, the amount of written data with GSSA is as low as that with the baseline allocation, and it is 72% lower than that with the MPP + FSP allocation. This is because, GSSA utilizes GSA and SSA blocks to significantly prevent the empty data (pages) from being programmed. Note that, the baseline allocation can also program empty data since the write requests (i.e., 4KB) can be smaller than a page size (16KB in our cases). Regarding the erase operation, Figure 14b demonstrates a similar trend as the amount of written data. Therefore, we mainly use the number of erases as the spent-lifetime indicator.

Allocation Overhead: Figure 15c shows the page allocation overheads under GSSA, the baseline and the MPP + FSP allocation schemes. Note that the baseline and the MPP + FSP allocation are, respectively, the 1-page and 8-page cases shown in Algorithm 3. On average, the allocation overhead of GSSA is smaller than both these of the baseline and the MPP + FSP allocation scheme. This is because GSSA performs exact number of page programs at a time; as a result, fewer allocations are performed and each allocation contains a minimal amount of empty data. To justify that, Figure 17 shows the execution latency distribution[11] of allocation on the Ex9 workload across three allocation scheme. In the figure, though the baseline allocation requires smaller latencies, the aggregated allocation overhead is higher than GSSA.

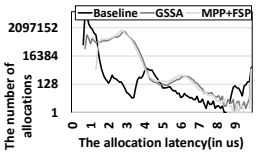


Fig. 17. The allocation latency distribution under Ex9 workload.

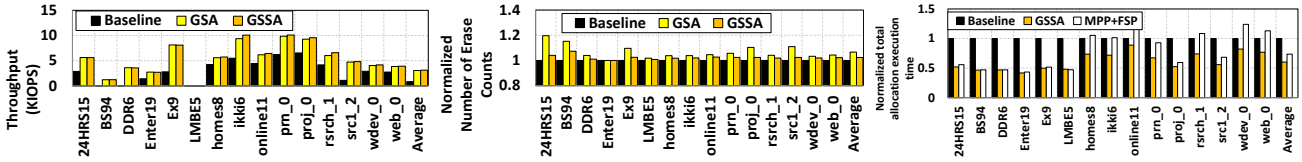
Combined Comparison: Figure 16c shows the combined comparison of performance and lifetime across all the allocation schemes tested in this work. Clearly, an "ideal" allocation scheme would achieve the highest throughput with the minimum lifetime consumption, and any allocation scheme mimicking such an ideal scheme should try to approach the upper left corner in this figure. It is clear that GSSA is the approach that comes closest to that corner, while the other allocation schemes are far away from the corner.

2) Contributions of GSA and SSA Employed by GSSA: To understand the individual performance/lifetime contributions of GSA and SSA employed in GSSA, Figures 15a and 15b plot the throughput and erase count comparisons, respectively, across the baseline, GSA, and GSSA allocation. As can be seen from Figure 15b, GSSA can achieve a lower spent-lifetime compared to GSA. This is because, the SSA blocks in GSSA can perform exact granularity program operations with a minimal amount of empty data. In contrast, GSA can only perform exact granularity programs when there are available supplemental writes. On the other hand, in terms of performance, as plotted in Figure 15a, the throughput of GSA and GSSA are similar since the GSSA optimization mainly targets lifetime.

3) Comparison against Other Proposals: Figures 16a and 16b plot, respectively, the performance and spent-lifetime comparison between the GSSA, dynamic page allocation and SLC cache.

Dynamic Page Allocation Proposals: Dynamic page allocations (DPA) algorithms [36], [37] have been proposed to improve the write throughput of static page allocation [35] (used by the baseline and GSSA) by redirecting the write requests from heavy-loaded chips to lightly-loaded chips. As a result, the written data can be programmed to NAND chips within a shorter period. However, DPA proposals cannot work

[11]We only measure the latencies on the allocation function (shown in Algorithm 3) execution on X86 machine.

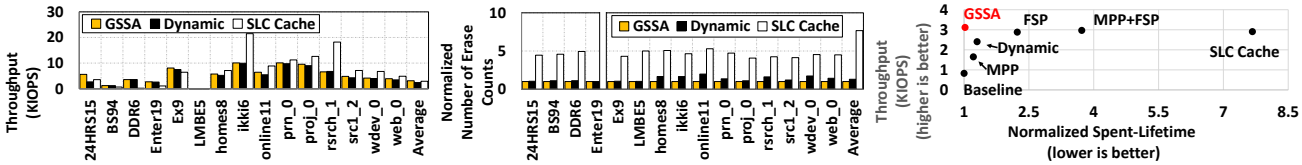


(a) Performance comparison against only generalized scramble allocation.

(b) Performance comparison against only generalized scramble allocation.

(c) Computation overhead comparison across the baseline, GSSA and the MPP + FSP schemes.

Fig. 15. Comparisons.



(a) Performance comparison against Dynamic page allocation and SLC Cache.

(b) Spent-lifetime comparison against Dynamic page allocation and SLC Cache.

(c) Performance and spent-lifetime comparison of various schemes.

Fig. 16. Comparisons.

well on the FSP-enabled NAND chips since they are not aware of the different granularity program operations.

To utilize the FSP and compare against our GSSA, our DPA implementation tries to accumulate written data till all data can be written by one (FSP) program operation. Then, such program operation is performed on the flash chips with less valid data to minimize the future garbage collection invocations, thereby minimizing the spent-lifetime.

As shown in Figure 16a , the performance of DPA cannot beat our GSSA since DPA is mainly optimized for write performance with potential read performance degradation. Specifically, the baseline static page allocation can ensure that the data are stripped across all chips; hence, the address-contiguous reads can be always serviced by multiple chips simultaneously, thereby sustaining good read performance. In contrast, DPA cannot ensure such property; so, the overall performance is lower compared to GSSA. Regarding to the lifetime (shown in Figure 16b), due to the "sync" operations issued by host, DPA *cannot* always accumulate enough data before programming. As a result, on average, DPA consumes more lifetime compared to GSSA.

SLC Cache Implementation: To improve the SSD read/write performance and achieve better storage benchmark scores, current 3D SSD manufacturers typically configure a small number of 3D NAND blocks to SLC mode [9]. By doing so, these SLC-mode blocks can achieve very low read/write latencies compared to QLC-baseline blocks; but, this comes with reduced overall SSD capacity. This implementation can, in principle, mitigate our targeted problem. Specifically, the small writes can be cached in the SLC-mode blocks before writing to other QLC-baseline blocks; so, the performance can be higher. In our implementation, we configure 4% of QLC blocks into SLC mode, which means the overall capacity is reduced to 97% of the original capacity.

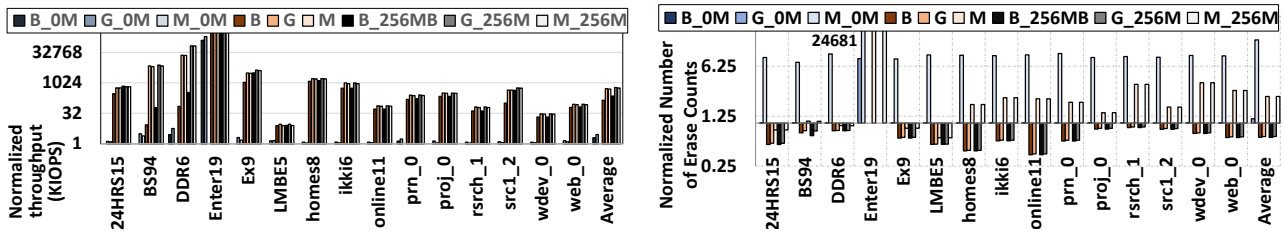
As shown in Figure 16a, the SLC cache can outperform our GSSA under many workloads, such as rsrch_1 and ikki6. This is because these workloads have small working sets, which can completely fit into the SLC-mode blocks; therefore, these workloads can enjoy the SLC-provided performance all the time. However, for the other large workloads, our GSSA outperforms the SLC cache, due to the frequent garbage collections caused by the small number of SLC-mode blocks and the reduced overall capacity. Regarding the lifetime (shown in Figure 16b), the SLC cache can significantly degrade the NAND chip lifetime. This is because the SLC cache implementation reduces the total capacity of the SLC-mode blocks to 1/4, which in turn causes more GC operations. Hence, these SLC-mode blocks will be worn-out much faster than the other (QLC-baseline) blocks.

Compared to these two proposals, our GSSA can improve performance without extensively consuming the SSD lifetime since GSSA can program the exact amount of written data via the GSA and SSA blocks, where the combined comparison can be found in Figure 16c.

4) Sensitivity to DRAM Capacity: In this subsection, we quantitatively show that large DRAM buffer cannot address the mentioned performance/lifetime problem. Figures 18a and 18b plot, respectively, the performance and spent-lifetime comparisons across the baseline, MPP + FSP, and GSSA, under three different DRAM capacities (0MB, 64MB, and 256MB). Both comparisons point to a similar trend, where the throughputs and spent-lifetimes of all three tested schemes increase significantly when the DRAM capacity is increased from 0MB to 64MB. However, the throughput values and spent-lifetimes only slightly improve when the DRAM capacity is further increased to 256MB. This is due to the "sync" operations that came with the writes, where the data in the DRAM buffer are forced to commit to the NAND chips. Thus, we can conclude that increased DRAM capacity cannot significantly improve performance. Therefore, a new resource allocation scheme, like GSSA, is required to address such performance/lifetime problem.

VII. Related Work

3D NAND-Aware Allocation Proposals: Emerging 3D NAND block-page architectures require new page allocation algorithms [46]–[49] to address the 3D NAND-specific problems, such as the additional disturbance from the vertical and horizontal directions. Chang et al. [47] proposed a page



(a) Throughput (Normalized to min(B_0M, M_0M)). (b) Spent-lifetime (Normalized to B_0M).

Fig. 18. Comparisons across the baseline, GSSA and the MPP + FSP allocations under different DRAM capacities (B: Baseline, G: GSSA, M: MPP + FSP).

allocation scheme, that programs the written data to the word-lines with no programmed word-lines around. As a result, the programmed data are free from the program disturbances coming from all directions. Wang et al. [48] proposed to evenly program data into different blocks. Thus, all blocks become free of the high-temperature induced disturbances. Chen et al. [49] investigated an allocation scheme that utilizes the asymmetric program latencies across layers, where the hot and cold data are programmed to the faster upper and the slower lower layers, respectively. We want to emphasize that all these mentioned allocation schemes consider only the 3D NAND architecture itself, and they are *not* aware of the different types of program operations enabled by 3D NAND. In contrast, GSSA not only considers 3D NAND architecture but also utilizes the various program operations brought by 3D NAND. As a result, it can improve the performance without excessively consuming SSD lifetime.

SSD-Internal Parallelism-Aware Proposals: Since the SSD architecture employs multiple channels, chips, dies, and planes, write requests can be executed simultaneously across chips to improve the overall throughput. Hu et al. [35] proposed to exploit chip-level parallelism, the interleave-die, and multi-plane commands, to improve performance. In comparison, Tavakkol et al. [36] adopted a workload-aware mechanism to evenly distribute the number of write requests across chips to balance the loads of the chips, enhancing the overall performance. However, due to the increased 3D NAND die density, 3D NAND-based SSDs consist of much fewer dies compared to their 2D counterparts; as a result, a lower die-level (or chip-level) parallelism can be utilized. Therefore, GSSA adopts the FSP operation and the proposed three strategies to address the 3D NAND performance degradation without excessively consuming the SSD lifetime.

VIII. Conclusion

While utilizing advanced program operations can significantly improve the performance of the 3D NAND based SSDs, the large granularity of such program operations can cause a lifetime degradation problem, where a small write is amplified to a large one with a significant fraction of empty data. Motivated by this observation, in this work, we propose and evaluate GSSA, a novel disturbance-aware resource allocation scheme, which takes into account the 3D NAND architecture specifics and exploits the diverse program operations enabled by the 3D NAND. Specifically, multiple allocation blocks per chip are used to service different granularity write requests at the same time. For the non-trivial write granularities, which cannot be serviced easily by the baseline program or other variations, we use GSA (generalized scramble allocation) and SSA (specialized scramble allocation) blocks to service them. By adopting GSSA, we can find the best location and the corresponding program operations to achieve high performance without excessively consuming precious SSD lifetime. The extensive experimental results show that the proposed GSSA achieves the performance of the MPP + FSP granularity (best performance) allocation scheme and, at the same time, the spent-lifetime of the baseline (best lifetime) scheme.

Acknowledgment

This research is supported by NSF grants 1629915, 1629129, 1931531, 2008398, 1822923 and 1908793, and a grant from Intel. Jung is supported by NRF 2016R1C1B2015312, DOE DE-AC02-05CH 11231, KAIST Start-Up Grant (G01190015), KAIST IDEC, ETRI 20RS1100, and Samsung Electronics (G01200447/G01200368). Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

References

[1] J.-W. Im, W.-P. Jeong, D.-H. Kim, S.-W. Nam, D.-K. Shim, M.-H. Choi, H.-J. Yoon, D.-H. Kim, Y.-S. Kim, H.-W. Park, and others, "7.2 A 128gb 3b/cell V-NAND flash memory with 1gb/s I/O rate," in *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE, 2015.

[2] D. Kang, W. Jeong, C. Kim, D. H. Kim, Y. S. Cho, K. T. Kang, J. Ryu, K. M. Kang, S. Lee, W. Kim, H. Lee, J. Yu, N. Choi, D. S. Jang, J. D. Ihm, D. Kim, Y. S. Min, M. S. Kim, A. S. Park, J. I. Son, I. M. Kim, P. Kwak, B. K. Jung, D. S. Lee, H. Kim, H. J. Yang, D. S. Byeon, K. T. Park, K. H. Kyung, and J. H. Choi, "7.1 256gb 3b/cell V-NAND flash memory with 48 stacked WL layers," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan. 2016.

[3] C. Kim, J. H. Cho, W. Jeong, I. h. Park, H. W. Park, D. H. Kim, D. Kang, S. Lee, J. S. Lee, W. Kim, J. Park, Y. l. Ahn, J. Lee, J. h. Lee, S. Kim, H. J. Yoon, J. Yu, N. Choi, Y. Kwon, N. Kim, H. Jang, J. Park, S. Song, Y. Park, J. Bang, S. Hong, B. Jeong, H. J. Kim, C. Lee, Y. S. Min, I. Lee, I. M. Kim, S. H. Kim, D. Yoon, K. S. Kim, Y. Choi, M. Kim, H. Kim, P. Kwak, J. D. Ihm, D. S. Byeon, J. y. Lee, K. T. Park, and K. h. Kyung, "11.4 A 512gb 3b/cell 64-stacked WL 3d V-NAND flash memory," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2017.

[4] S. Lee, C. Kim, M. Kim, S. m. Joe, J. Jang, S. Kim, K. Lee, J. Kim, J. Park, H. J. Lee, M. Kim, S. Lee, S. Lee, J. Bang, D. Shin, H. Jang, D. Lee, N. Kim, J. Jo, J. Park, S. Park, Y. Rho, Y. Park, H. j. Kim, C. A. Lee, C. Yu, Y. Min, M. Kim, K. Kim, S. Moon, H. Kim, Y. Choi, Y. Ryu, J. Choi, M. Lee, J. Kim, G. S. Choo, J. D. Lim, D. S. Byeon, K. Song, K. T. Park, and K. h. Kyung, "A 1tb 4b/cell 64-stacked-wl 3d NAND flash memory with 12mb/s program throughput," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018.

[5] D. Kang, M. Kim, S. C. Jeon, W. Jung, J. Park, G. Choo, D. Shim, A. Kavala, S. Kim, K. Kang, J. Lee, K. Ko, H. Park, B. Min, C. Yu, S. Yun, N. Kim, Y. Jung, S. Seo, S. Kim, M. K. Lee, J. Park, J. C. Kim, Y. S. Cha, K. Kim, Y. Jo, H. Kim, Y. Choi, J. Byun, J. Park, K. Kim, T. Kwon, Y. Min, C. Yoon, Y. Kim, D. Kwak, E. Lee, W. Hahn, K. Kim, K. Kim, E. Yoon, W. Kim, I. Lee, S. h. Moon, J. Ihm, D. S. Byeon, K. Song, S. Hwang, and K. H. Kyung, "13.4 a 512gb 3-bit/cell 3d 6th-generation v-nand flash memory with 82mb/s write throughput and 1.2gb/s interface," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, Feb 2019.

[6] R. Yamashita, S. Magia, T. Higuchi, K. Yoneya, T. Yamamura, H. Mizukoshi, S. Zaitsu, M. Yamashita, S. Toyama, N. Kamae, J. Lee, S. Chen, J. Tao, W. Mak, X. Zhang, Y. Yu, Y. Utsunomiya, Y. Kato, M. Sakai, M. Matsumoto, H. Chibvongodze, N. Ookuma, H. Yabe, S. Taigor, R. Samineni, T. Kodama, Y. Kamata, Y. Namai, J. Huynh, S. E. Wang, Y. He, T. Pham, V. Saraf, A. Petkar, M. Watanabe, K. Hayashi, P. Swarnkar, H. Miwa, A. Pradhan, S. Dey, D. Dwibedy, T. Xavier, M. Balaga, S. Agarwal, S. Kulkarni, Z. Papasaheb, S. Deora, P. Hong, M. Wei, G. Balakrishnan, T. Ariki, K. Verma, C. Siau, Y. Dong, C. H. Lu, T. Miwa, and F. Moogat, "11.1 a 512gb 3b/cell flash memory on 64-word-line-layer bics technology," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017.

[7] H. Maejima, K. Kanda, S. Fujimura, T. Takagiwa, S. Ozawa, J. Sato, Y. Shindo, M. Sato, N. Kanagawa, J. Musha, S. Inoue, K. Sakurai, N. Morozumi, R. Fukuda, Y. Shimizu, T. Hashimoto, X. Li, Y. Shimizu, K. Abe, T. Yasufuku, T. Minamoto, H. Yoshihara, T. Yamashita, K. Satou, T. Sugimoto, F. Kono, M. Abe, T. Hashiguchi, M. Kojima, Y. Suematsu, T. Shimizu, A. Imamoto, N. Kobayashi, M. Miakashi, K. Yamaguchi, S. Bushnaq, H. Haibi, M. Ogawa, Y. Ochi, K. Kubota, T. Wakui, D. He, W. Wang, H. Minagawa, T. Nishiuchi, H. Nguyen, K. H. Kim, K. Cheah, Y. Koh, F. Lu, V. Ramachandra, S. Rajendra, S. Choi, K. Payak, N. Raghunathan, S. Georgakis, H. Sugawara, S. Lee, T. Futatsuyama, K. Hosono, N. Shibata, T. Hisada, T. Kaneko, and H. Nakamura, "A 512gb 3b/cell 3d flash memory on a 96-word-line-layer technology," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018.

[8] N. Shibata, K. Kanda, T. Shimizu, J. Nakai, O. Nagao, N. Kobayashi, M. Miakashi, Y. Nagadomi, T. Nakano, T. Kawabe, T. Shibuya, M. Sako, K. Yanagidaira, T. Hashimoto, H. Date, M. Sato, T. Nakagawa, H. Takamoto, J. Musha, T. Minamoto, M. Uda, D. Nakamura, K. Sakurai, T. Yamashita, J. Zhou, R. Tachibana, T. Takagiwa, T. Sugimoto, M. Ogawa, Y. Ochi, K. Kawaguchi, M. Kojima, T. Ogawa, T. Hashiguchi, R. Fukuda, M. Masuda, K. Kawakami, T. Someya, Y. Kajitani, Y. Matsumoto, N. Morozumi, J. Sato, N. Raghunathan, Y. L. Koh, S. Chen, J. Lee, H. Nasu, H. Sugawara, K. Hosono, T. Hisada, T. Kaneko, and H. Nakamura, "13.1 a 1.33tb 4-bit/cell 3d-flash memory on a 96-word-line-layer technology," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, Feb 2019.

[9] C. Siau, K. Kim, S. Lee, K. Isobe, N. Shibata, K. Verma, T. Ariki, J. Li, J. Yuh, A. Amarnath, Q. Nguyen, O. Kwon, S. Jeong, H. Li, H. Hsu, T. Tseng, S. Choi, S. Darne, P. Anantula, A. Yap, H. Chibvongodze, H. Miwa, M. Yamashita, M. Watanabe, K. Hayashi, Y. Kato, T. Miwa, J. Y. Kang, M. Okumura, N. Ookuma, M. Balaga, V. Ramachandra, A. Matsuda, S. Kulkani, R. Rachineni, P. K. Manjunath, M. Takehara, A. Pai, S. Rajendra, T. Hisada, R. Fukuda, N. Tokiwa, K. Kawaguchi, M. Yamaoka, H. Komai, T. Minamoto, M. Unno, S. Ozawa, H. Nakamura, T. Hishida, Y. Kajitani, and L. Lin, "13.5 a 512gb 3-bit/cell 3d flash memory on 128-wordline-layer with 132mb/s write performance featuring circuit-under-array technology," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, Feb 2019.

[10] T. Tanaka, M. Helm, T. Vali, R. Ghodsi, K. Kawai, J. K. Park, S. Yamada, F. Pan, Y. Einaga, A. Ghalam, T. Tanzawa, J. Guo, T. Ichikawa, E. Yu, S. Tamada, T. Manabe, J. Kishimoto, Y. Oikawa, Y. Takashima, H. Kuge, M. Morooka, A. Mohammadzadeh, J. Kang, J. Tsai, E. Sirizotti, E. Lee, L. Vu, Y. Liu, H. Choi, K. Cheon, D. Song, D. Shin, J. H. Yun, M. Piccardi, K. F. Chan, Y. Luthra, D. Srinivasan, S. Deshmukh, K. Kavalipurapu, D. Nguyen, G. Gallo, S. Ramprasad, M. Luo, Q. Tang, M. Incarnati, A. Macerola, L. Pilolli, L. D. Santis, M. Rossini, V. Moschiano, G. Santin, B. Tronca, H. Lee, V. Patel, T. Pekny, A. Yip, N. Prabhu, P. Sule, T. Bemalkhedkar, K. Upadhyayula, and C. Jaramillo, "7.7 a 768gb 3b/cell 3d-floating-gate nand flash memory," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016.

[11] C. Liu, J. Kotra, M. Jung, and M. Kandemir, "PEN: Design and evaluation of partial-erase for 3d nand-based high density ssds," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, 2018.

[12] ——, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *Proceedings of the Twenty-fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, 2019.

[13] "Dramexchange website," https://www.dramexchange.com/.

[14] "Toshiba and western digital readying 128-layer 3d nand flash," https://www.techpowerup.com/253373/toshiba-and-western-digital-readying-128-layer-3d-nand-flash.

[15] R. Micheloni, L. Crippa, and R. Ravasio, "Double page programming system and method," https://www.google.tl/patents/US20070030732, Feb. 8 2007, uS Patent 20070030732.

[16] R. A. Cernea, L. Pham, F. Moogat, S. Chan, B. Le, Y. Li, S. Tsao, T. Y. Tseng, K. Nguyen, J. Li, J. Hu, J. H. Yuh, C. Hsu, F. Zhang, T. Kamei, H. Nasu, P. Kliza, K. Htoo, J. Lutze, Y. Dong, M. Higashitani, J. Yang, H. S. Lin, V. Sakhamuri, A. Li, F. Pan, S. Yadala, S. Taigor, K. Pradhan, J. Lan, J. Chan, T. Abe, Y. Fukuda, H. Mukai, K. Kawakami, C. Liang, T. Ip, S. F. Chang, J. Lakshmipathi, S. Huynh, D. Pantelakis, M. Mofidi, and K. Quader, "A 34 mb/s mlc write throughput 16 gb NAND with all bit line architecture on 56 nm technology," *IEEE Journal of Solid-State Circuits*, Jan 2009.

[17] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[18] "3D XPoint technology," https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[19] A. M. Rino Micheloni, Luca Crippa, *Inside NAND Flash Memory*. Springer Netherlands, 2010.

[20] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, "Amber: Enabling precise full-system simulation with detailed modeling of all ssd resources," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. IEEE Press.

[21] W. Choi, M. Jung, and M. Kandemir, "Invalid data-aware coding to enhance the read performance of high-density flash memories," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018.

[22] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "HIOS: A host interface i/o scheduler for solid state disks," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014.

[23] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014.

[24] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "Graphssd: Graph semantics aware ssd," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19, 2019.

[25] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[26] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme," *ACM Transactions on Embedded Computing Systems*, Mar. 2010.

[27] "SATA protocol," https://sata-io.org/.

[28] "NVMe storage protocol," https://nvmexpress.org/.

[29] J. Zhang, G. Park, M. M. Shihab, D. Donofrio, J. Shalf, and M. Jung, "OpenNVM: An open-sourced fpga-based nvm controller for low level memory characterization," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, Oct 2015.

[30] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE.

[31] K. S. Shim, E. S. Choi, S. W. Jung, S. H. Kim, H. S. Yoo, K. S. Jeon, H. S. Joo, J. S. Oh, Y. S. Jang, K. J. Park, S. M. Choi, S. B.

Lee, J. D. Koh, K. H. Lee, J. Y. Lee, S. H. Oh, S. H. Pyi, G. S. Cho, S. K. Park, J. W. Kim, S. K. Lee, and S. J. Hong, "Inherent issues and challenges of program disturbance of 3d nand flash cell," in *2012 4th IEEE International Memory Workshop*, May 2012.

[32] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Improving 3D nand flash memory lifetime by tolerating early retention loss and process variation," *SIGMETRICS Perform. Eval. Rev.*, 2018.

[33] Q. Xiong, F. Wu, Z. Lu, Y. Zhu, Y. Zhou, Y. Chu, C. Xie, and P. Huang, "Characterizing 3d floating gate NAND flash," in *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '17 Abstracts. ACM, 2017.

[34] "Scsi storage interfaces," http://www.t10.org/.

[35] 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 *Proceedings of the international conference on Supercomputing (SC)*, 2011.

[36] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad, "Unleashing the potentials of dynamism for page allocation strategies in SSDs," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '14, 2014.

[37] 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.*, Jun. 2016.

[38] M. Kwon, J. Zhang, G. Park, W. Choi, D. Donofrio, J. Shalf, M. Kandemir, and M. Jung, "TraceTracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2017.

[39] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory ming: Experimental Analysis, Exploits, and Mitigation Techniques," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2017.

[40] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2019.

[41] N. Shahidi, M. Arjomand, M. Jung, M. T. Kandemir, C. R. Das, and A. Sivasubramaniam, "Exploring the potentials of parallel garbage collection in ssds for enterprise storage systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, 2016.

[42] Y. H. Chang, J. W. Hsieh, and T. W. Kuo, "Improving flash wear-leveling by proactively moving static data," Jan 2010.

[43] M. Murugan and D. H. C. Du, "Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2011.

[44] J. Li, X. Xu, X. Peng, and J. Liao, "Pattern-based write scheduling and read balance-oriented wear-leveling for solid state drivers," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2019.

[45] F. Chen, M. Yang, Y. Chang, and T. Kuo, "Pwl: A progressive wear leveling to minimize data migration overheads for nand flash devices," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015.

[46] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019.

[47] 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 *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '13. IEEE Press, 2013.

[48] Y. Wang, M. Zhang, L. Dong, and X. Yang, "A thermal-aware physical space allocation strategy for 3D flash memory storage systems," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. ACM, 2016.

[49] S.-H. Chen, Y.-T. Chen, H.-W. Wei, and W.-K. Shih, "Boosting the performance of 3d charge trap nand flash with asymmetric feature process size characteristic," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17, 2017.