Are Superpages Super-fast? Distilling Flash Blocks to Unify Flash Pages of a Superpage in an SSD

Shih-Hung Tseng[†], Tseng-Yi Chen[†], and Ming-Chang Yang[‡]
Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan[†]
Department of Computer Science and Engineering, The Chinese University of Hong Kong[‡]
hardaway.tseng@gmail.com, tychen@g.ncu.edu.tw, mcyang@cse.cuhk.edu.hk

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00055

***Abstract*—This work discovers a flash memory performance issue resulting from flash superpages organization. Because of process variation, each flash page has its own read/write performance. If a slow page is grouped with a fast page in a superpage unit, computer systems with solid-state drives (SSD) receive a poor performance result. In this work, we prove the existence of this issue by conducting a series of experiments on a real SSD platform. To resolve this issue, we characterize flash memory chips to find hints to organize super-fast superpages in SSDs. By the tips, this work develops a process-variation check scheme (PV Check) that can group a superpage with an optimized performance at runtime with low overheads. According to our experiments, the PV Check scheme has encouraged results in performance improvement. Compared with a traditional method, our work can decrease the extra program and erase latency by 16.61% and 34.55%, respectively.**

Keywords-super page, flash memory, solid-state drive, process variation

I. Introduction

As a solid-state drive (SSD) consists of multiple independent channels and chips, data can be written to different channels and chips in parallel so as to realize high internal parallelism. Internal parallelism becomes one of NAND flash memory-base SSDs to support high I/O throughput. Multiple blocks from different chips are organized as a superblock to simplify data management on a modern SSD with high parallelism. Moreover, SSD provides multi-plane (MP) com- mands to read/write from/to a superblock in parallel. When an MP command is issued to each plane, this command is completed after the issued operation is finished on all planes. However, because of process variation, flash blocks have different characteristics and performances. Therefore, while a slow block is organized with fast blocks as a superblock, the slow block decreases this superblocks's performance. Extra latency is called the performance gap between the slowest and fastest blocks in a superblock. This study conducts a series of experiments on real flash memory chips to observe the phenomena of extra latency. Our observation also motivates us to design a strategy for organizing superblocks with minimal extra program/erase latency.

NAND flash memory-based SSDs have been widely applied to different environments (i.e., cloud services, data centers, and deep learning platforms [1]) because of their fast I/O performance and high internal parallelism. Although NAND flash memory has some bad characteristics (e.g., limited endurance, write amplification, and cell disturbance [5]), many research works have proposed well-designed garbage collection mechanisms [8], [29], wear-leveling strategies [6], [21], and pattern-aware data placement [9], [12]. But besides these bad characteristics, high parallelism might raise the difficulty in data management. Many previous works present superblock management schemes in the flash translation layer [13], [36] to ease the problem. A superblock is organized by multiple blocks from different flash memory chips. However, owing to MP commands, the performance of a fast block is slowed down by a slow block in the same superblock. Specifically, flash blocks have different performances and characteristics because of process variation. Therefore, the most important problem is how to organize the blocks with strongly similar characteristics in one superblock.

As process variation naturally happens on transistors (i.e., different lengths, widths, and thicknesses) during the process of fabrication, flash blocks on different chips and planes have different performances and characteristics. Previously, many researchers have conducted a series of experiments on real flash memory chips to reveal the fact of process variation on flash memory chips. Based on this fact, many solutions have been proposed in dynamically adjusting read and writing voltage for each word-lines [16], [32] and applying different refresh management strategies for retention loss to improve flash memory lifetime [10], [19]. On the other hand, some researchers explore the similarity between adjacent physical word-line layers and flash blocks. When physical word-lines and flash blocks in a row have strong similarities, a system management scheme can explore the characteristics of a few leading blocks and apply the configurations collected from monitoring the leading blocks to other flash blocks for in-creasing flash memory access performance [17] and extending flash memory lifetime [33]. However, in these previous studies, they only observe the process variation and similarity on the same flash chips. In flash memory, a superblock is organized by blocks from different chips. Therefore, observing process variation on different chips would be a key point to prove that randomly organizing superblocks cannot obtain the optimal superblock performance.

In this study, we conduct a series of experiments to explore the process variation on different flash memory chips. According to our experiments, each word-lines and flash block has its own program and erase latency. In other words, one flash memory chip has fast and slow flash blocks. If we organize a slow block and multiple fast blocks in a superblock, the slow block degrades the superblock performance.

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00055

Therefore, we propose a practical and near-optimal solution to organize superblocks on demand. Our proposed scheme can effectively minimize the extra latency. Based on our experimental results on real flash memory chips, our solution can shorten the extra program and erase latency by 16.61% and 34.55%, respectively. The contributions of this study can be summarized as follows.

- This work first defines extra latency in superblocks and conduct a series of experiments on 24 real flash memory chips to explore the existence of the extra latency in superblock.
- This work proposes eight directions, including one local optimal solution, to organize superblocks and evaluate their capabilities on real flash memory chips.
- To realize a practical solution, this study proposes a method, namely QSTR-MED, that significantly reduces the computing overhead and provides comparable performance with the local optimal solution.

Paper organization. Section II introduces SSD and flash memory knowledge. In Section III, our optimized goal, extra latency is observed from experiments on real flash memory chips. Section IV elaborates on all possible solutions and their performance, and Section V presents our proposed scheme in detail. To show the effect of our proposed scheme on the extra latency reduction, Section VI includes the experimental settings and results. Section VII talks about our related works. Finally, concluding remarks are summarized in Section VIII.

II. Solid-State Drive (SSD)

Solid-state drives have been widely applied to servers and personal computers nowadays because of their shock resistance, low power consumption, and high I/O performance. Moreover, owing to the advanced technologies (e.g., triple-level cell (TLC) and quad-level cell (QLC)) and manufacturing process (e.g., 3D integrated circuit), the price of an SSD swiftly dropped to be affordable for most applications and platforms. Generally, an SSD consists of a controller, a DRAM (or SRAM) memory space, and a NAND flash memory array.

A controller, i.e., flash translation layer (FTL), hides the characteristics of NAND flash memory to integrate with a host system as a normal block-based storage drive. In addition to the controller, an SSD contains many engines for specific applications. For example, an error correction code (ECC) engine detects and corrects error bits when a page is read from NAND flash memory, and an encrypt/decrypt engine enhances data security by encrypting data in the storage device. The DRAM space of an SSD not only temporarily stores read/write data as a data buffer but also caches some information to address data location in the SSD. The most important part of an SSD is the NAND flash memory array which is the main storage space. Generally, the NAND flash memory array comprises multiple channels, each consisting of at least two flash memory chips. Because each channel has its own channel controller and data bus, data can be written to different channels in parallel. Accordingly, high parallelism has become one of the nice features of an SSD. After introducing the basic architecture of an SSD, this section includes more information about the NAND flash memory structure in Section II-A and the high internal parallelism of SSDs in Section II-B.

A. Flash Memory Technology

NAND flash memory is a floating-gate-based non-volatile memory. A floating gate transistor holds an electrical charge to store data persistently. Generally, a number of floating-gate transistors are connected in series as a data unit (i.e., word-line) to avert complex circuit layout, thereby increasing data storage density. Figure 1 illustrates the modern flash memory architecture.

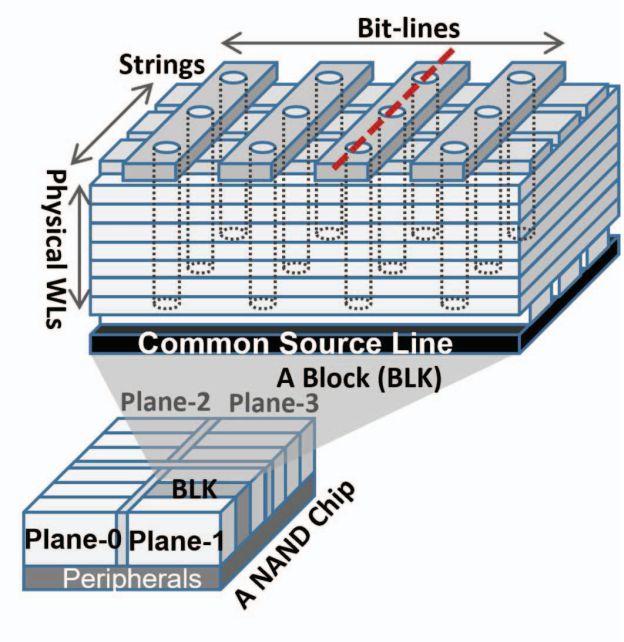

Figure 1. 3D NAND Flash Memory Organization.

In Figure 1, a NAND flash memory chip is composed of several planes (e.g., four planes), each of which has many flash blocks (BLK) that is the basic unit for erasing (ERS). Moreover, a block consists of many physical word- lines (PWL) and several strings (STR). A physical word-line and a string produce a logical word-line (LWL or WL), which is the basic unit for programming. Figure 1 details a NAND flash blocks. In this example, a block has 96 physical word- lines (i.e., 3D NAND flash memory with 96 layers) and four strings; thus, the number of logical word-line (WL) is 384 (i.e., from 0 to 383). A programming process respectively applies programming and bypassing voltage to one selected and other unchosen physical word-lines; moreover, one of the strings is also activated. Consequently, the program process can inject charges to the cells on the selected word-line whose string is switched on. Each word-line connects many bit lines (BL). The number of bit lines depends on the width (i.e., the number of bits) of a data page. In terms of an 18KB page (i.e., 16KB

for user data and 2KB for spare area), there are 147,456 (18 × 1024 (bytes) × 8 (bits)) bit-lines controlled by word-lines and strings. Furthermore, a logical word-line stores one to four pages based on cell types. For instance, a logical word-line has three data pages, i.e., least significant bit (LSB), central significant bit (CSB), and most significant bit (MSB) pages, when the flash memory cell is a triple-level cell (TLC). Specifically, the LSB page is composed of the least significant bit of memory cells within a logical word- line. Like the LSB page, both the CSB and MSB pages serve the purpose of encapsulating the central and most significant bits of memory cells located within the same logical word- line, respectively. A page is the basic data unit to be read. For the support of high access speed, some multi-plane (MP) commands, including multi-plane page read, multi-plane word-line program, and multi-plane block erase, can be issued to operate pages, word-lines, and blocks on different planes in parallel. Note that a multi-plane command is completed when the issued operation is finished on targeted pages, word-lines, or blocks.

B. *SSD Internal Parallelism*

As aforementioned, SSD is a data storage device with high parallelism. In an SSD, all flash memory chips work independently. To increase the storage throughput in a single operation, the modern SSD controller provides a way to access the NAND flash memory array in parallel. Specifically, the SSD controller groups multiple blocks from different planes and chips as a superblock (SB). In other words, a superblock is assembled by picking one of the blocks from each plane in different chips of channels. In a superblock, physical word- lines labeled the same word-line number are bonded as a super word-line. In a super word-line, pages with the same page type (i.e., LSB, CSB, and MSB) are organized as a superpage. Figure 2 illustrates the concepts of the superblock, super word-line (Super WL), and superpage. In this organization, word-lines in the same super word-line are programmed simultaneously, and pages in the same superpage can be read in parallel. For example, as depicted in Figure 2, a superblock is formed by aggregating Block 0 (defined as the block with an offset of 0) on Chip 0, Block 25 on Chip 1, and Block 25 on Chip m. A multi-plane command is reported as a completed command when the issued command is done on all planes. However, this feature might increase a single I/O latency when fast and slow word-lines and pages are grouped in the same super word-line and superpage, respectively.

III. Process Variation And Similarity

Because of the advanced manufacturing process, the process node size gradually scaled down. Although small transistors (i.e., small process node size) are high performance, space- and power-efficiency, it also results in serious process variation. Process variation is the natural variation in the transistor's attributes (i.e., width, length, and oxide thickness) during the fabrication of integrated circuit. Transistors with small process node sizes (i.e., <65nm) would have pronounced process variation. Moreover, transistors have varied performance owing to process variation.

To alleviate the impact of the process variation on flash memory performance, many previous studies have been proposed to improve read/writing performance [10], [15], [27], enhancing flash memory reliability [16], [31], and extending lifetime [19]. For example, Wang et al. [32] propose a scheme to make process variation transparent to the host system. Then, they deploy the proposed scheme on an open-channel SSD to allocate appropriate storage spaces for user data to reduce uncorrectable error bits. Moreover, Luo et al. characterize layer-to-layer variation in flash memory and propose four techniques to prolong flash memory lifetime. Although the process variation is truly occurring in the flash memory fabrication process, it does not mean that all flash pages have varied performance.

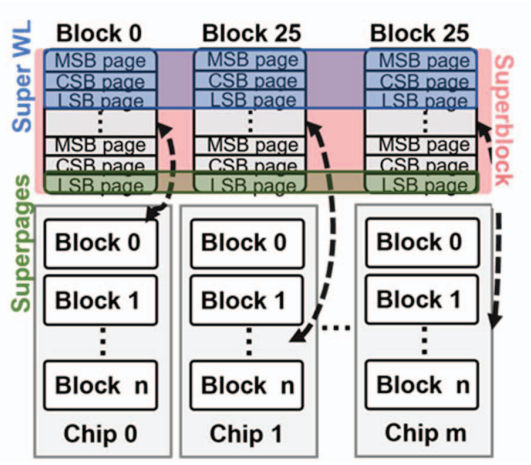

Figure 2. Superblock, super word-line, and superpage.

Recently, NAND flash memory vendor has turned to the development of a three-dimensional structure because the technology of semiconductor fabrication has always encountered limitations on down-scaling. Specifically, the margin between bit-growth and cost in the 2D planar NAND flash memory becomes very narrow and has appeared dis-economies [34]. Figure 3 presents the organization of 3D NAND flash memory.

The etching process of 3D NAND flash memory fabrication makes bit-lines channel physically and vertically, running through the word-line layers. Due to the etching process, the aperture of bit-line channels becomes different in each word-line layer. As illustrated in Figure 3, the pipe of the V- shape channel, which creates bigger cells in the top layers and smaller cells in the bottom layer, is formed. It results in cells in the same word-line layer having a similar size. The different apertures of cells between word-line (WL) layers influence the characteristics of performance and reliability. To redeem the diversified characteristics between WL layers, NAND vendors practically organize WL layers into several groups and dynamically apply different operating parameters such as erase, program, verify voltage and pulse timing to each WL layer group. By this approach, the characteristics and reliability between WL layers will get closer to each other. Moreover, process similarity has also resulted from this approach.

By exploiting the benefit of the process similarity, many excellent previous research studies have also been proposed in speeding up read/write performance [17], [28] and prolonging flash memory lifetime [33]. Specifically, Shim et al. [28] observe that the horizon layer of flash memory has strong process similarity. Because of the strong process similarity, they carefully reuse some parameters obtained from the leading WL to quickly access the rest of the WLs on the same horizontal layer. Additionally, Yen et al. [33] observe the leading block to determine the timing of bad block retirement so as to minimize the probability of uncorrectable error occurrence and extend flash memory lifetime. To sum up, different flash memory planes might encounter process variation; however, some flash memory blocks or the word-lines on the same horizontal layer also have strong process similarity.

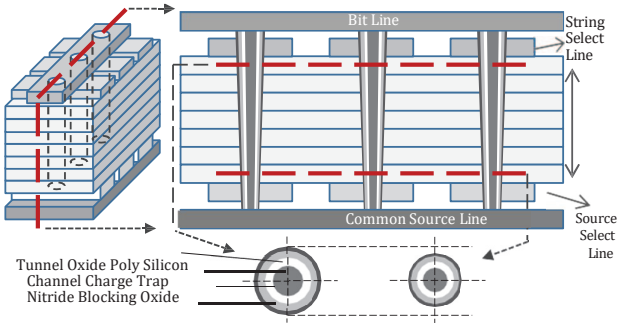

Figure 3. The organization of 3D NAND flash memory

A. Motivation: Super-fast Superpages

Because of the process variation, each block and page have its own characteristics, including the number of error bits, programming, reading, and erasing latencies. However, some blocks have strong process similarity due to the organization of 3D flash memory. As aforementioned, an SSD has the property of high parallelism because it has special units, namely superblock, super word-line, and superpages, for data access. However, a superblock containing varied performance blocks would result in extra latency. The concept of extra latency is the performance gap between the fastest and slowest block in the superblock. Figure 4 illustrates the concept of extra latency in programming and erasing.

As aforementioned, superblock and super word-line operations require that the issued command need to be done on all blocks in the superblock and logical word-lines in the super word-line due to the constraint of MP commands. Therefore, some fast blocks or word-lines need to wait for other slow blocks or word-lines to finish their task. In Figure 4, the block situated on plane 1 (P1) exhibits the most extended erase latency, while the block on plane 2 (P2) demonstrates the shortest erase latency. Consequently, the additional erase latency is quantified as the difference between the erase latency of the block on P1 and the block on P2. Furthermore, when we talk about "extra program latency," we mean the time difference between the fastest and slowest word-line programming within the memory. In a superblock, the "extra program latency" is the total time difference for all the word-lines in that superblock. Thus, the storage performance seriously degrades if a flash memory controller cannot organize a superblock well.

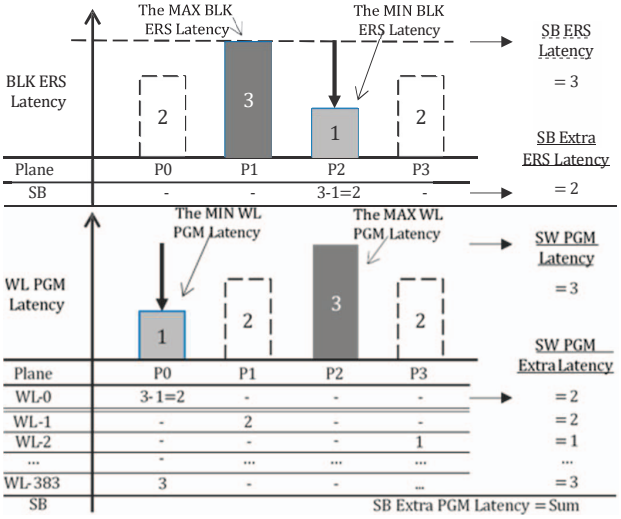

Figure 4. Extra latency of erasing superblock and programming super word-line. The BLK ERS latency pertains to the block erase duration, while the WL PGM latency corresponds to the word-line programming time.

To validate our assumptions, we conducted experiments on actual flash memory chips to gather data on block erase and word-line programming latencies. Further details regarding our experimental methodology can be found in Section VI. Figure 5 illustrates the block erase and word-line programming latency.

The top section of Figure 5 reveals variations in the erasing performance of blocks from different chips (or chip enable (CE)). In this study, we collected data from 1603 blocks, sourced from two distinct chips, each equipped with four planes. Notably, the left and right portions of Figure 5 correspond to different chips. The different colors on the graph represent different planes, with the x-axis denoting block numbers and the y-axis representing block erase latency. A consistent, unchanging line signifies those multiple consecutive blocks have same block erase latency, while deviations or drops in the line indicate blocks with shorter erasing latency. Conversely, spikes points indicate slower-performing blocks.

Our analysis demonstrates that block erasing latency varies between blocks on these two chips. Additionally, we examined word-lines in the lower section of Figure 5. The x-axis corresponds to word-line numbers, and the y-axis represents word- line programming latency for four blocks on four different planes. In this experiment, we collected data from two different chips, focusing on groups of four blocks in sequence. From this observation, we discerned that the programming latency trend remains consistent across all word-lines within the same chip. Nevertheless, different chips exhibit distinct patterns in word-line program latency. Notably, between word-line 150 and word-line 200, we could

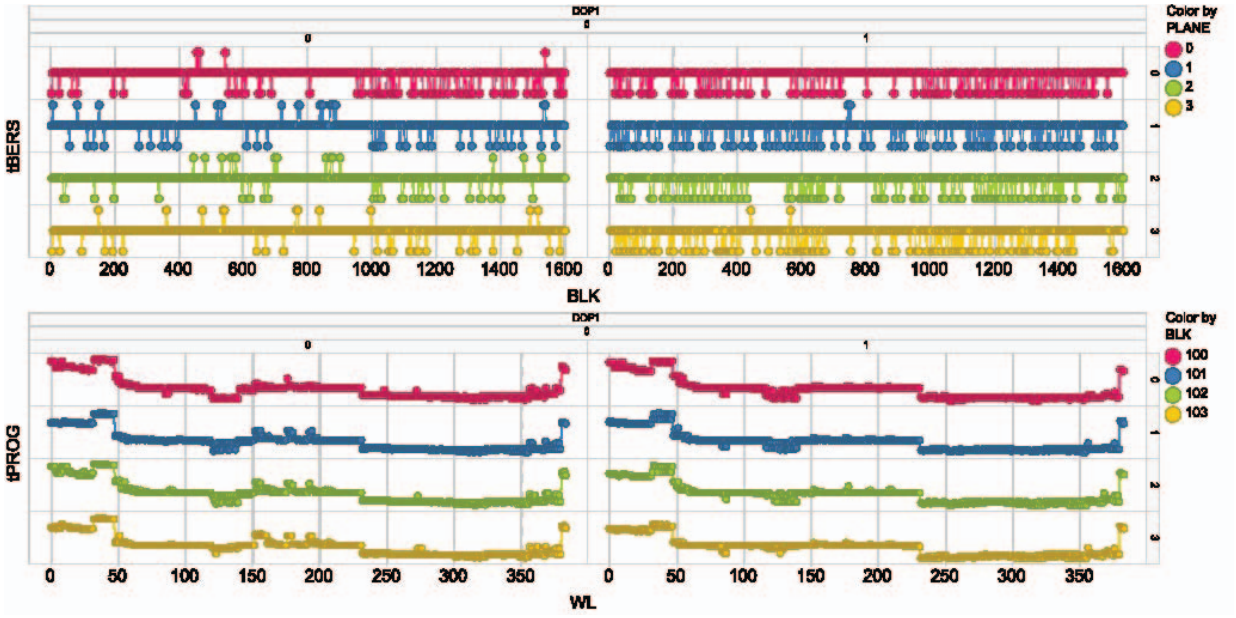

Figure 5. Collected latency of erasing block (top) and programming word-line (bottom) of different blocks. tBERs represents the erase latency for each block. Collected from two different chips on a double die NAND flash memory package. tPROG denotes the program latency for each word-line.

observe performance variations between the two different chips. Furthermore, it is noteworthy to highlight that previous research [24] has corroborated that the variability in endurance among blocks across different chips is 6.69 times higher than the variability among blocks within the same chip. Through our observation, we identified process variation across different chips, but blocks within the same chip demonstrated process similarity. Furthermore, some blocks on different chips exhibited similar performance, as indicated in Figure 5.

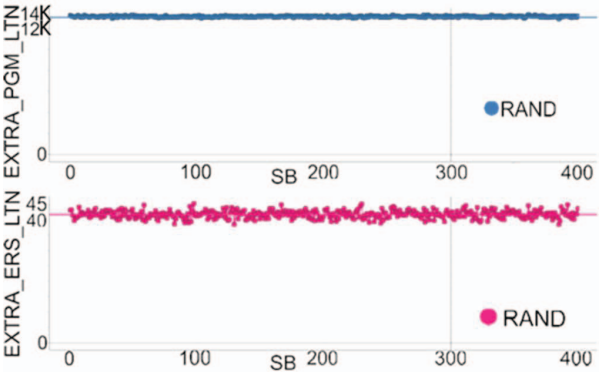

Figure 6. The extra program (top)/erase (bottom) latency by random method.

As a result, grouping the same performance block in a superblock becomes the most efficient way to boost flash memory performance because the extra latency should be as short as possible. To show the importance of superblock organization, we conduct experiments to show the amount of extra latency when we randomly group blocks from different chips as a superblock. As illustrated in Figure 6, the y-axis is the average latency in the unit of a microsecond, and the x-axis is the block number. Note that the superblock program latency is the sum of the extra program latency of all word-lines in this superblock. Obviously, the results show the significant extra latency, 13,084.17μs for programming and 41.71μs for erasing on average. Furthermore, we also observe the amount of extra latency under different program/erase (P/E) cycles by the random method. *The goal of this work is to unify blocks and word-line performance in a superblock so as to minimize the extra latency.*

IV. Characterization and Findings

A. Potential Organization Methods

As our goal is to minimize the extra latency of each superblock, our design principle should be grouping blocks with similar characterizations from different chips as a superblock. To be guided by this principle, we propose some possible directions for organizing superblocks, as follows.

1) Sequential assembly.: This direction is inspired by the process similarity of 3D NAND flash memory architecture. This method simply comes from the thought that blocks with the same sequence number on different chips might have process similarity. Specifically, *SB 1* is organized by the first block from each chip, and *SB 2* is assembled by the second block from each chip. This approach is a simple and intuitive direction. However, this method would not be optimal method if there is no process similarity between all blocks with the same sequence number on different chips.

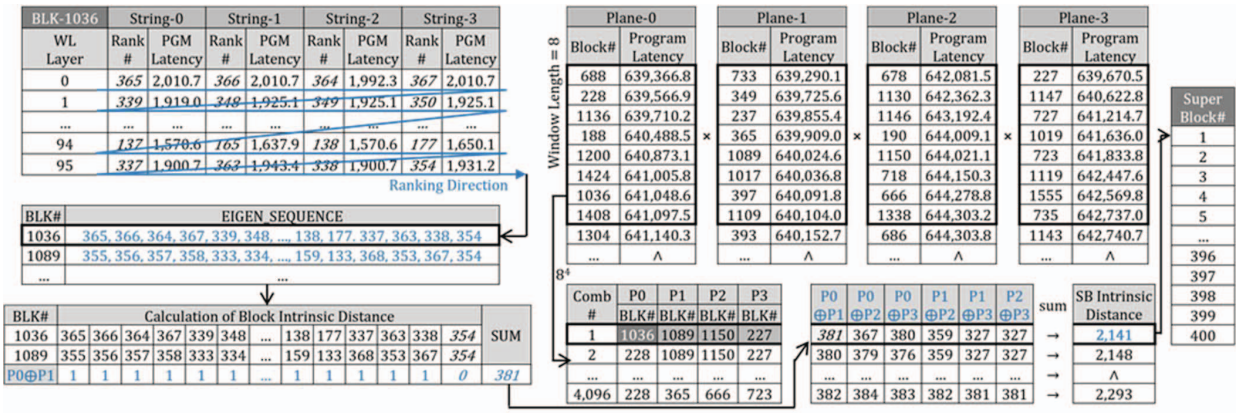

Figure 7. The LWL-rank assembly.

2) Erase latency assembly.: This direction is also intuitive and simple. In this direction, we assume that two blocks have strong process similarity if these two blocks have similar erase latency. In this approach, we iteratively assemble blocks from fast to slow on different chips to organize multiple superblocks.

3) Program latency assembly.: In the previous approach, block erase latency is an indicator to show the similarity. This direction utilizes program latency as the indicator of process similarity. However, a flash block does not have the parameter of program latency. Therefore, we just calculate the sum of the program latency of all word-lines in one block to represent the program latency of a block. After obtaining the program latency of blocks, we follow the same process with the erase latency assembly to iteratively select blocks from fast to slow on different chips to group superblocks. However, in this method, blocks are sorted by their program latency.

4) Local optimal assembly.: This direction is to find out the local optimal solution for superblock organization. In this direction, we set the local window size to 8. With the windows size, this direction assembles superblocks with minimal pro- gram latency. To achieve the optimal assembly, this direction utilizes a brute force method to check every combination for minimal program latency. This experiment has 4,096 (i.e., 8^4) combinations for four flash memory chips. Moreover, because every memory chip can assemble 400 super blocks in each P/E cycle, this method needs to check 1,638,400 (4,096 × 400) combinations. Although this local optimal solution is impractical, this data can be the ground reference indicating the best result.

5) LWL-rank assembly.: This direction firstly introduces the metric of block distance (i.e., D (i)) to evaluate the similarity of two different blocks. The distance between two blocks is defined in Equation 1. Firstly, this method ranks all logical word-lines (LWL) in a block according to their program latency. Then, this method selects two blocks (i.e., *i* and *j*) from two different flash memory chips. To calculate the distance of these two blocks, we just simply sequentially compare the rank of each word-line in these two blocks. The distance between these two blocks adds nothing if the compared word-lines have the same rank. Otherwise, the distance is increased by one.

$$D(i) = \sum_{j=i+1}^{C_N} \sum_{wl=0}^{LWL_N} SIM(i,j,wl), \quad (1)$$

where

$$SIM(i,j,wl) = \begin{cases} 1, Block_{i,wl} \neq Block_{j,wl} \\ 0. Block_{i,wl} = Block_{j,wl} \end{cases},$$

where C_N denotes the number of chips, and LWL_N is the number of logical word-lines. Note that *SIM(i, j, wl)* equals one if the compared word-lines in these two blocks have the same rank. Figure 7 presents this method in detail.

As illustrated in Figure 7, this method also ranks blocks in chips at the first step according to blocks' program latency. Then, it selects one combination in the window size (e.g., 8) to calculate the distance. In this round, we compare two blocks (i.e., Blocks 1036 and 1089). This method ranks all logical word-lines according to their program latency in the compared block. After rank, this method stores the rank information in a vector for the calculation of block distance. As illustrated in Figure 7, if the compared word-lines in these two blocks have the same rank (i.e., the last logical word-line ranked 354), the distance would add nothing (i.e., zero). Otherwise, the sum of distance would be increased by one. After calculating the distance of all combinations, we can select the combination with the shortest distance as our superblock.

6) PWL-rank assembly.: This direction is similar to the LWL-rank assembly. However, it ranks physical word-lines. In the previous direction, we rank all logical word-lines in a block. Therefore, the word-line would be ranked from 0 to 383 (96 × 4). In this direction, we only rank physical word-lines, and each string has its own physical word-line rank. Because the number of physical word-line layers is 96, each string ranks its physical word-lines from 0 to 95 according to the physical word-line program latency. After ranking, the

PWL-rank assembly also converts the rank information to a vector and utilizes the same process to calculate the distance.

7) STR-rank assembly.: This method is also similar to the previous two solutions. In this approach, we just change the ranking strategy to the string (STR) orientation. By the new ranking method (i.e., STR-based ranking), the logical word- lines would be ranked from one to four because there are four strings in one block. After ranking, we follow the same process by Equation 1 to calculate the distance and find the combination with minimal distance.

8) STR-median assembly.: The last approach is the simple version of the previous method. This method is also based on STR-based ranking; however, we only assign two rank numbers (i.e., 0 and 1). If the string on a physical word- line is top two fast, we assign zero to the physical word-line. Otherwise, one is assigned to the physical word-line. Because only one bit is assigned to each physical word-line, we can simply do an XOR operation to calculate the distance.

B. Results and Findings

This work proposes eight directions to optimize superblock and superpage performance. In this section, we conduct a series of real platform experiments to obtain the results of characterization. These results also guide us in a direction to develop a systematic and practical solution to organize superblocks. The results of characterization are derived from the experiments with 24 NAND flash memory chips under P/E cycles from 0 to 3,000 at step 200. In the experiments, our baseline solution is random assembly. We summarize the experimental results in Table I.

TABLE I. THE RESULTS OF THESE EIGHT DIRECTIONS.

Method	PGM LTN ↓ (Avg.)	Imp. %
SEQUENTIAL	1,367.57 μs	10.45%
ERS-LTN	1,118.35 μs	8.55%
PGM-LTN	1,356.38 μs	10.37%
OPTIMAL (8)	**2,550.73 μs**	**19.49%**
LWL-RANK (8)	1,845.64 μs	14.11%
PWL-RANK (8)	2,036.86 μs	15.57%
STR-RANK (8)	**2,390.05 μs**	**18.27%**
STR-MED (4)	**2,189.94 μs**	**16.74%**

As presented in Table I, the best solution is the local optimal method that can reduce the extra program latency by 19.49%. However, this solution is impractical because of its high time complexity in organizing superblocks. STR-rank assembly is the closest solution to the local optimal. This is because STR- rank can provide a modest amount of information to organize superblocks. On the other hand, LWL-rank and PWL-rank might provide too much information for superblock organization since they classify PWL and LWL into more levels (e.g., rank information from 0 to 95 (PWL) and 380 (LWL)). Although STR-rank assembly is also impractical because of its high time complexity, it also gives us a hint about how the approach can preserve the information of block similarity[1]. Moreover, a large window size (e.g., 8) results in high time complexity because the number of combinations is too large. The results under different window size settings should be observed to find the most appropriate window size setting. We collect the results of the STR-RANK approach under different window sizes.

TABLE II. STR-RANK WITH DIFFERENT WINDOW SIZES.

Method	PGM LTN ↓ (Avg.)	Imp. %
STR-RANK (8)	2,390.05 μs	18.27%
STR-RANK (6)	2,361.06 μs	18.05%
STR-RANK (4)	**2,279.14 μs**	**17.42%**
STR-RANK (2)	1,965.78 μs	15.02%

As shown in Table II, the reduction of program latency increases as the window size enlarges. However, a large window size results in huge computing overhead to organize the superblock with strong block similarity. With the considerations of time complexity and program latency reduction, assigning the window size as four is the best solution. Moreover, STR-MED can further reduce the time and space complexity because it only utilizes one bit to represent the rank information of strings and a simple bitwise operation to calculate the distance of two block pairs. However, we still cannot apply the STR-rank or STR-MED solutions to the flash memory controller because we need to consume plenty of time to check all combinations for superblock organization. When flash memory has four planes, and the window size is four, the number of combinations is 256. Additionally, we should check the six block pairs' distance for each combination. The total number of distance checks is 1,536.

V. PROCESS SIMILARITY CHECK SCHEME

A. System Overview

The main goal of our proposed scheme is to reduce the number of combinations checks to a reasonable number. To achieve our goal, our scheme, QSTR-MED, collaborates with three major components for gathering information, assembling superblocks, and allocating storage spaces for data in a performance-oriented manner. Figure 8 illustrates the system architecture of our proposed QSTR-MED.

As presented in Figure 8, the proposed QSTR-MED accumulates the current block's program latency in LTN SUM and converts its word-line program latency to a block eigenvector formed by a series of bit zero and one. After all word-lines in a block are programmed, the information-gathering method triggers the updater to update the sorted program latency list and block's eigenvalue space (see Section V-B for more information). The sorted program latency list keeps the blocks of each chip in the order of block's program latency. By the information of the sorted list and block's eigenvalue space, the superblock can be organized on demand.

[1]Blocks have similar performance

Specifically, if a fast superblock is required, our superblock assembly selects the first four blocks in the sorted list from different chips for the similarity check. Otherwise, the assembly chooses the last four blocks in the list from different chips to do a similarity check. By applying our assembly, all superblocks, including fast and slow superblocks, have minimal extra latency (see Section V-C in detail). Because our assembly can organize fast and slow superblocks on demand, the QSTR-MED provides a data management manner to boost storage system performance by guiding written data to fast and slow superblocks (refer to Section V-D for more details). With these three components, the QSTR-MED becomes practical for flash memory storage.

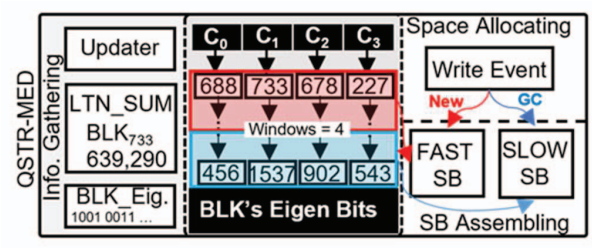

Figure 8. The proposed QSTR-MED scheme.

B. Gathering: Similarity Data

Our gathering process collects similarity information at runtime during program operations. As the QSTR-MED is inspired by the STR-MED, data for similarity check include the block's program latency and the rank of the program latency of word-lines in a block. Figure 9 depicts the process of gathering similarity data in detail.

BLK-733	STR-0	STR-1	STR-2	STR-3
PWL	WL PGM Latency			
0	1917.0	1898.6	1898.6	1898.6
1	1898.6	1898.6	1898.6	1898.6
...	...	...	...	...
94	1579.1	1646.6	1579.1	1579.1
95	1898.6	1910.8	1880.1	1910.8

Sum all = BLK PGM LTN= 639,290.1

Eigen_sequence			
1	0	0	1
0	0	1	1
...	...	...	...
0	1	0	1
0	1	0	1

1001 0011 ... 0101 0101

Figure 9. Process of gathering similarity information.

The gathering process includes two data structures (i.e., latency table and eigenvectors) to maintain the collected similarity information. The latency table keeps block's all physical word-line program latency information that is updated after a physical word-line is programmed. When all physical word- lines on the same physical word-line layer are programmed, the gathering process utilizes one bit to rank the word-lines on this physical WL layer. On one physical WL layer, the fastest two physical WLs are marked bit zero. Otherwise, the other WLs would be labeled bit one. If many physical word-lines have the same performance, the gathering process sequentially assigns bits zero to the first two word-lines. A physical WL generates an eigenvector formed by bits zero and one. When all physical WL layers in one block are programmed, the eigenvalue sequence is generated by sequentially joining all eigenvectors. In other words, an eigenvalue sequence and blocks' program latency information are obtained after a block is completely programmed. The eigenvalue sequence is information for similarity comparison, and the block's program latency guides the free block to the correct position in the residing chip's sorted list. Note that we do not allocate the spaces of the latency table and eigenvectors for every block, only for open blocks.

C. Assembling: Superblock Similarity

After the block's eigenvalue sequence is maintained, the QSTR-MED utilizes this information to organize the same characteristics of blocks from different chips as a superblock in a practical manner. The nice features of our QSTR-MED are computing complexity reduction and superblock organization on demand. Figure 10 illustrates the block selection of superblock organization.

PLANE-0		PLANE-1		PLANE-2		PLANE-3	
BLK#	PGM_LTN	BLK#	PGM_LTN	BLK#	PGM_LTN	BLK#	PGM_LTN
688	639,366.8	733	639,290.1	678	642,081.5	227	639,670.5
228	639,566.9	349	639,725.6	1130	642,362.3	1147	640,622.8
1136	639,710.2	237	639,855.4	1146	643,192.4	727	641,214.7
188	640,488.5	365	639,909.0	190	644,009.1	1019	641,636.0
1200	640,873.1	1089	640,024.6	1150	644,021.1	723	641,833.8
...	∧	...	∧	...	∧	...	∧
868	650,124.9	85	649,708.9	1530	654,378.4	151	652,040.6
544	650,135.5	9	649,783.0	482	654,378.4	771	652,090.1
464	651,125.1	1541	650,036.7	1378	654,594.9	539	652,275.7
456	651,912.7	1537	650,067.6	902	655,083.4	543	652,566.4

(fast)

BLK#	Eigen-SEQ	BLK#	Eigen-SEQ	BLK#	Eigen-SEQ	BLK#	Eigen-SEQ
688	1010 1010 ...	733 (fast-est)	1001 0011 ...	678	1001 0011 ...	227	0011 0011 ...
228	0101 0011 ...			1130	0110 0101 ...	1147	0011 1001 ...
1136	1010 0011 ...			1146	0011 0101 ...	727	0011 0011 ...
188	1100 0011 ...			190	0101 0101 ...	1019	0101 0110...

(Slow)

868	0011 1010 ...	85	0011 0110 ...	902 (slow-est)	0011 1100 ...	151	0011 0011 ...
544	0011 1010...	9	0011 0011 ...			771	0011 0011 ...
464	0011 1010 ...	1541	0011 0011 ...			539	0110 0011 ...
456	0011 0011 ...	1537	0011 0110 ...			543	0110 0110 ...

Figure 10. The block selection for superblock assembly.

In our assembly, the first thing is to determine what kind of superblocks should be organized. With the request of fast superblock organization, the assembly first selects the fastest block among the head blocks in all sorted lists. The fastest block would be organized in the superblock, and then the assembly selects the first four blocks in other sorted lists. As shown in Figure 10, the fastest block (i.e., *BLK 733*) is organized in the fast superblock at the first step. Next, the assembly chooses the fastest four blocks from other chips as candidates for superblock organization. On the other hand, we follow a similar process to organize a slow superblock, but we need to find the slowest block and the slowest four blocks from other chips.

After block selection, we have one selected block and multiple candidates. Accordingly, the assembly only needs to

check the similarity between the selected block and candidates. In the similarity check process, the assembly loads the eigenvalue sequences of the selected blocks and one candidate and then operates an XOR operation. The distance of similarity is the number of bits 1 in the result from the XOR operation. Because this operation is simple, we can design a circuit to accelerate this comparison and accumulation. After the calculation, we can select blocks with minimal similarity distance from each chip to assemble as a superblock. Because this superblock has a strong similarity between blocks and the selected block, the extra latency can be effectively reduced. Moreover, our QSTR-MED does not compute the similarity distance between all block pairs; instead, we only calculate the distance between the selected one and other blocks. As a result, the latency for combination checks is significantly decreased.

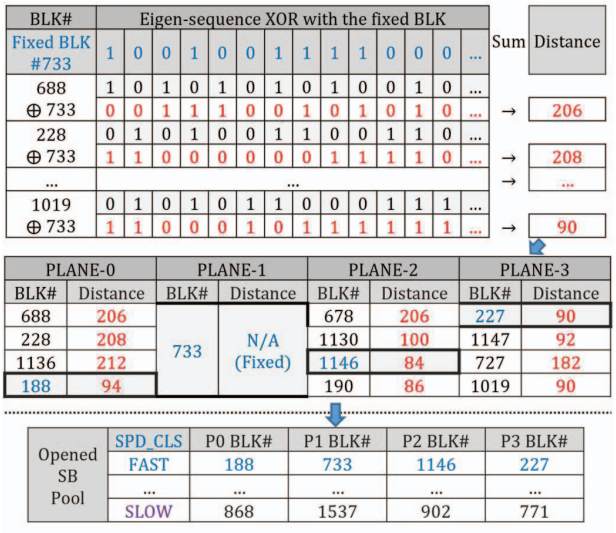

Figure 11. Checking similarity in a superblock

D. Allocating: Function-based Placement

Because the QSTR-MED can organize superblocks on demand, we can guide the written data to different superblock for further boosting storage performance. Although the QSTR- MED achieves our design goal, "minimizing the extra latency in superblocks", superblocks still can be classified into fast and slow superblocks. To allocate the appropriate superblock for written data, the QSTR-MED detects the types of written data. Suppose that data is written by user requests from the host system side. Our QSTR-MED allocates a fast superblock for the written data. Suppose some system developers want to optimize storage system performance further. In that case, they can write small random data to a high-speed superpage and large batch data to a slow superpage in the fast superblock. On the other hand, if garbage collection processes generate written data, a slow superblock can receive the written data to avoid a slow superblock decreasing flash memory storage performance from the aspect of host systems.

VI. Performance Evaluation

A. Experimental Environment

The purpose of this section is to evaluate the effects of the proposed QSTR-MED on the extra latency optimization and computing overhead reduction. All experimental data are collected from a real platform. This platform is composed of four SMI SM2259XT SATA 3.0 controllers, 24 3D NAND flash memory chips, and a KSON chamber. Table III summarizes our experimental environment.

TABLE III. HARDWARE AND SOFTWARE PLATFORMS

Item	Part Name
SSD Controller	SMI SM2259XT SATA 3.0 x 4
NAND Flash	SKH H25BFT8B3M8R (DDP) x 4 SKH H25BFT8D4M8R (QDP) x 4
Chamber	KSON TS-F5T-150 x 1
IPC	KSON THS-Z200 x 1
PC	INTEL i5-3570, DDR-16GB, SSD 1TB
Visual Analysis	TIBICO Spotfire 6.5.0

In our experiments, a chip has four planes, each of which is composed of four planes. A plane consists of 954 blocks. In a block, there are four strings and 96 physical WL layers; thus, it has 1,152 pages. To speed up our experiments, we evenly separated chips into two groups. One group collects data for the first 1,600 blocks; another group is formed from the last 1,600 blocks. The experimental setting of NAND flash memory in detail is presented in Table IV. Moreover, we collect the superpage and superblock latency at some predefined timings. Specifically, our predefined timings are under different P/E cycles from 1 to 3,000 at step 100 and 6 high-temperature data retention (HTDR).

TABLE IV. TESTING SETTINGS OF NAND FLASH MEMORY

PKG	CH	CE	# of CHIP	Block Range
DDP #1-1	0	0/1	2	4 ,1,603
DDP #1-2	2	0/1	2	1,604 ,3,275
DDP #2-1	0	0/1	2	4 ,1,603
DDP #2-2	2	0/1	2	1,604 ,3,275
QDP #1-1	0	0/1/2/3	4	4 ,1,603
QDP #1-2	2	0/1/2/3	4	1,604 ,3,203
QDP #2-1	0	0/1/2/3	4	4 ,1,603
QDP #2-2	2	0/1/2/3	4	1,604 ,3,203

B. Evaluation Results

1) Performance Improvement: As the performance of some strategies has been presented in Section IV-B, our experiments only contain the results of four strategies, including random as the baseline, local optimal, STR-MED, and our QSTR-MED, organizing superblock. The goal of this study is to minimize the extra latency in programming and erasing operations. Therefore, performance improvement is the most important metric for the evaluation of the QSTR-MED capability. Table V presents the extra program and erase latency on average. In Table V, although the optimal solution has minimal extra program and erase latency, it is not a feasible solution for flash memory. On the contrary, our

QSTR-MED has comparable performance with the optimal solution, and it is a practical solution. In Table V, the erase latency is smaller than the program latency because the program latency is the sum of the extra latency of all word-lines in a block. For example, consider a block comprising 384 word-lines, where the average extra program latency per word-line amounts to 34μs. The cumulative extra program latency (Extra PGM LTN) for this superblock would thus be 13,056μs (calculated as 384 × 34μs).

TABLE V. THE RESULTS OF EXTRA PROGRAM AND ERASE LATENCY

Methods	Extra PGM LTN	Extra ERS LTN
Random	13,084.17 μs	41.71 μs
Sequential	11,716.60 μs	40.12 μs
Optimal	10,533.44 μs	22.65 μs
QSTR-MED(4)	10,911.53 μs	25.10 μs
STR-MED(4)	10,894.23 μs	24.97 μs

Figure 12 clearly demonstrates the impact of QSTR-MED on performance enhancement. In comparison to the baseline method (random), our QSTR-MED reduces program (PGM) and erase (ERS) latencies by 16.61% and 59.82%, respectively. Additionally, it introduces only a 378.09µs delay when compared to the optimal solution. These results highlight the substantial reduction in extra program and erase latencies achieved by QSTR-MED.

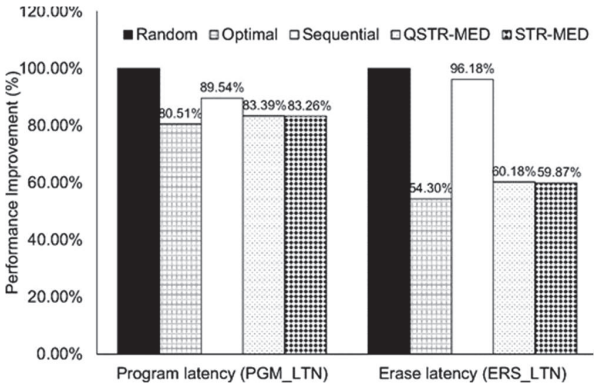

Figure 12. Performance improvement in the program and erase latency.

When compared to the sequential solution[2] commonly implemented in modern SSDs, QSTR-MED exhibits a 7% improvement in extra program latency (Extra PGM LTN), but a significant 37.43% reduction in extra erase latency (Extra ERS LTN). Furthermore, QSTR-MED serves as a practical counterpart to STR-MED. Figure 15 showcases superblocks organized by STR-MED and QSTR-MED, revealing that their capabilities and performance in reducing extra latency are equivalent, with QSTR-MED being the practical choice.

Moreover, in Figure 13, the distributions of the extra program and erase latency can prove that our proposed QSTR-MED can effectively minimize the extra latency. In Figure 13, the X-axis is the extra latency, and the Y-axis denotes the number of superblocks. Therefore, when a solution can move the distribution to the left, this solution can effectively reduce the extra latency because many superblocks result in short extra latency. As illustrated in Figure 13, our QSTR-MED can move the distribution (i.e., program latency) to left.

[2]It is a super-page group with the same offset in different chips.

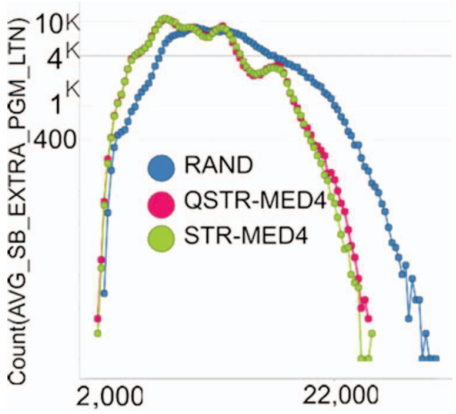

Figure 13. The distributions of the extra PGM LTN.

2) Computing overhead: As previously mentioned, QSTR-MED represents a practical iteration of STR-MED, primarily due to its substantial reduction in computing overhead associated with checking combinations. When the window size is four, STR-MED necessitates computing overhead to assess the similarity among 1,536 combinations. In contrast, QSTR- MED, at the same window size, only needs to evaluate the similarity between 12 pairs of blocks. This efficiency arises from QSTR-MED's selection of the fastest or slowest block as the reference, thus requiring the identification of other blocks with the strongest similarity from different chips for superblock organization.

QSTR-MED significantly reduces computing overhead by an impressive 99.22% when compared to STR-MED. Further- more, as depicted in Figure 14, the performance trends of STR-MED and QSTR-MED closely mirror each other. This demonstrates that QSTR-MED can achieve substantial performance improvements at a significantly lower cost compared to STR-MED.

C. Advanced analysis: Sensitivity of P/E cycles

While we have previously demonstrated the efficacy of QSTR-MED in improving extra program/erase latency, it is essential to assess its robustness under high failure rates when an SSD drive is subject to wear and tear. To conduct this evaluation, we collected QSTR-MED performance data under varying P/E (Program/Erase) cycles, as high P/E cycles tend to result in elevated bit error rates. In Figure 15, the X-axis represents P/E cycles ranging from 0 to 3000, while the Y-axis displays the average program and erase latency. As depicted in Figure 15, our QSTR-MED exhibits consistent program and erase latency, regardless of increasing P/E cycles. This

resilience stems from QSTR-MED's ability to consistently organize superblocks with minimal extra latency. In other words, our QSTR-MED stands as a robust solution that remains unaffected by high failure rates.

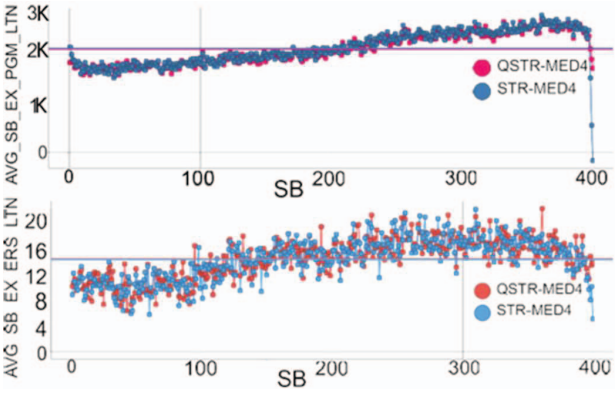

Figure 14. All superblocks improvement.

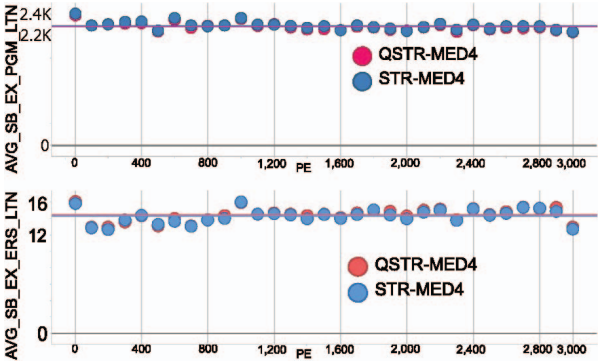

Figure 15. Performance improvement under different P/E cycles. Program (Top) and erase latency (Bottom).

D. Overhead analysis of QSTR-MED

1) Space overhead: Demonstrating the practicality of our proposed solution necessitates an analysis of its space and computing overheads. QSTR-MED's requirements are minimal, encompassing solely two pieces of information: the block program latency (BLK PGM LTN) and the eigen sequence per block. We accumulate word-line program latency onto block program latency after each word-line is programmed, necessitating only a single integer for this purpose. Additionally, representing whether a logical word-line is faster or slower than the average word-line program latency requires just one bit. Consequently, our space overhead can be succinctly described by Equation 2.

$$M_{footprint} = N_{blo} \quad \times (S_{PGM\_LTN} + S_{Eigen}) \qquad (2)$$

where N_{block} denotes the number of blocks in an SSD. Plus, S_{PGM_LTN} and S_{Eigen} refer to the memory sizes required to store information of the block program latency and the eigen sequence, respectively. Consider a block with 384 logical word-lines. In such a scenario, we require an integer (e.g., 4 bytes) to store block program latency information and 384 bits for the eigen sequence. Consequently, each block needs 52 bytes to accommodate its metadata for superblock organization. If we apply our QSTR-MED to a 1TB SSD comprising multiple 8MB blocks, our memory footprint amounts to 6.5MB. When juxtaposed with the working memory size of a typical 1TB SSD, the memory footprint of our QSTR-MED is exceptionally small.

2) Computing accelerator: While our QSTR-MED has already substantially reduced the computing overhead compared to STR-MED, there is potential for further performance enhancement. Notably, the most time-consuming tasks in QSTR-MED are generating the eigen sequence and calculating similarity distances. To optimize these processes, we can implement a comparator circuit during eigen sequence generation to compare current word-line program latency with the aver- age latency from the previous round. For similarity distance calculations, an XOR circuit and bit counter circuit can be employed to swiftly compute the distance between two blocks. These hardware circuits can further boost the performance of QSTR-MED, despite its already reduced computing overhead compared to STR-MED.

VII. Related Works

SSD Parallelism. As SSD is an electronic storage device composed of multiple flash memory chips, it has high internal parallelism by writing data to different flash memory chips simultaneously. Although the SSD internal parallelism brings fast data access, it also raises the difficulty in data management. To simplify the complexity of data management, previous studies [14], [18] propose FTL schemes for managing superblock and superpage data units. In superblock and super- page management, garbage collection [26] and flash memory' lifetime [30], [35] would be two critical issues. Therefore, new garbage collection designs and wear-leveling techniques have been proposed to tackle these two critical issues. Additionally, some SSD system designs utilize the property of internal parallelism to combine RAID system management [13], [36] to further enhance flash memory's reliability. However, these works only discuss data management schemes and applications (e.g., RAID). They have not considered and observed the extra latency resulting from blocks in a superblock. As process variation naturally occurs on a transistor during fabrication, each block in a superblock should have different characteristics.

Process Variation. Because of the small process node size resulting from the advanced manufacturing process, transistors have different characteristics (e.g., endurance [11] and performance [22]) caused by the varied width, length, and thickness. With the fact of the process variation, flash memory pages and blocks have different reliability (i.e., raw-bit error rate (RBER)) [20] and I/O performance [10], [15], [27]. To observe the influence of the process variation on flash memory performance, some excellent works conduct a series of experiments in real platforms [2], [3], [25]. Moreover,

previous research works present some feasible solutions to resolve endurance (or reliability) issue [16], [31] and varied I/O performance [10], [15], [27] resulted from the process variation. In addition to the in-drive solutions, the process variation also has been studied in the hardware circuit design [4] and the development of the host system interface [7].

Process Similarity. Fortunately, because of the spatial locality, some flash blocks or word-lines have a strong similarity. Specifically, logical word-lines on the same physical word- line layer [23] and the adjacent flash blocks [33] might have a strong similarity. By exploiting the benefits of similarity, flash management systems can set a forerunner for observation. Then, according to the observed characteristics, flash management systems can adjust the controller configuration to enhance flash memory lifetime [33] and performance [17], [28].

VIII. Concluding Remarks

This study conducted a series of experiments to reveal that arbitrarily organizing superblocks results in long extra latency. To minimize the extra latency, this study proposes eight directions, including a local optimal solution, for finding an organization method. Then, we collect the characterization results of these eight directions based on real-platform experiments. Unfortunately, the solutions with positive results are not practical for flash memory controllers because of the high computing overhead. To design a practical solution, this study develops the QSTR-MED inspired from one of eight directions. Because the QSTR-MED significantly reduces the computing overhead, it becomes a feasible solution to organize superblocks with minimal extra latency at runtime. Moreover, our QSTR-MED can minimize the extra program and erase latency in all superblocks.

Acknowledgment

This work is supported in part by National Science and Technology Council (Project Nos. 111-2221-E-008-058-MY3 and 111-2628-E-008-002-MY3) and The Research Grants Council of Hong Kong SAR (Project Nos. CUHK14210320 and CUHK14218522).

References

[1] E. K. Ardestani, C. Kim, S. J. Lee, L. Pan, J. Axboe, V. Rampersad, B. Agrawal, F. Yu, A. Yu, T. Le, H. Yuen, D. Mudigere, S. Juluri, A. Nanda, M. Wodekar, K. Nair, M. Naumov, C. Petersen, M. Smelyan- skiy, and V. Rao, "Supporting massive dlrm inference through software defined memory," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 302–312.

[2] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 1285–1290.

[3] 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)*, 2013, pp. 123–130.

[4] M.-F. Chang and S.-J. Shen, "A process variation tolerant embedded split-gate flash memory using pre-stable current sensing scheme," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 3, pp. 987–994, 2009.

[5] Y.-M. Chang, Y.-H. Chang, T.-W. Kuo, Y.-C. Li, and H.-P. Li, "Distur- bance relaxation for 3d flash memory," *IEEE Transactions on Comput- ers*, vol. 65, no. 5, pp. 1467–1483, 2016.

[6] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 212–217. [Online]. Available: https://doi.org/10.1145/1278480.1278533

[7] J. Chen, Y. Wang, A. C. Zhou, R. Mao, and T. Li, "Patch: Process-variation-resilient space allocation for open-channel ssd with 3d flash," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 216–221.

[8] T.-Y. Chen, Y.-H. Chang, Y.-H. Kuan, and Y.-M. Chang, "Virtualgc: Enabling erase-free garbage collection to upgrade the performance of rewritable slc nand flash memory," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[9] T.-Y. Chen, Y.-H. Chang, Y.-H. Kuan, M.-C. Yang, Y.-M. Chang, and P.-C. Hsiu, "Enhancing flash memory reliability by jointly considering write-back pattern and block endurance," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 5, aug 2018. [Online]. Available: https://doi.org/10.1145/3229192

[10] Y. Di, L. Shi, C. Gao, Q. Li, C. J. Xue, and K. Wu, "Minimizing retention induced refresh through exploiting process variation of flash memory," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 83–98, 2019.

[11] A. P. Ferreira, S. Bock, B. Childers, R. Melhem, and D. Mosse´, "Impact of process variation on endurance algorithms for wear-prone memories," in *2011 Design, Automation & Test in Europe*, 2011, pp. 1–6.

[12] J. Guo, D. Wang, Z. Shao, and Y. Chen, "Data-pattern-aware error prevention technique to improve system reliability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1433–1443, 2017.

[13] D. Hong, K. Ha, M. Ko, M. Chun, Y. Kim, S. Lee, and J. Kim, "Reparo: A fast raid recovery scheme for ultra-large ssds," *ACM Trans. Storage*, vol. 17, no. 3, aug 2021. [Online]. Available: https://doi.org/10.1145/3450977

[14] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "Superblock ftl: A superblock-based flash translation layer with a hybrid address translation scheme," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 4, apr 2010. [Online]. Available: https://doi.org/10.1145/1721695.1721706

[15] Q. Li, L. Shi, Y. Di, Y. Du, C. J. Xue, and E. H. Sha, "Exploiting process variation for read performance improvement on ldpc based flash memory storage systems," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 681–684.

[16] Q. Li, L. Shi, Y. Di, C. Gao, C. Ji, Y. Liang, and C. J. Xue, "Process variation aware read performance improvement for ldpc-based nand flash memory," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 310–321, 2020.

[17] Q. Li, M. Ye, Y. Cui, L. Shi, X. Li, T.-W. Kuo, and C. J. Xue, "Shaving retries with sentinels for fast read over high-density 3d flash," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 483–495.

[18] P.-K. Lin, M.-L. Chiao, and D.-W. Chang, "Improving flash translation layer performance by supporting large superblocks," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 642–650, 2010.

[19] 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," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, dec 2018. [Online]. Available: https://doi.org/10.1145/3224432

[20] R. Ma, F. Wu, Z. Lu, W. Zhong, Q. Wu, J. Wan, and C. Xie, "Blockhammer: Improving flash reliability by exploiting process variation aware proactive failure prediction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4563–4574, 2020.

[21] M. Murugan and D. 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)*, 2011, pp. 1–12.

[22] A. Mutlu and M. Rahman, "Statistical methods for the estimation of process variation effects on circuit operation," *IEEE Transactions on Electronics Packaging Manufacturing*, vol. 28, no. 4, pp. 364–375, 2005.

[23] S. Nie, Y. Zhang, W. Wu, and J. Yang, "Layer rber variation aware read performance optimization for 3d flash memories," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[24] Y. Pan, H. Zhang, M. Gong, and Z. Liu, "Process-variation effects on 3d tlc flash reliability: Characterization and mitigation scheme," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020, pp. 329–334.

[25] Y. Pan, H. Zhang, M. Gong, and Z. Liu, "Process-variation effects on 3d tlc flash reliability: Characterization and mitigation scheme," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020, pp. 329–334.

[26] 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 *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 561–572.

[27] L. Shi, Y. Di, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting process variation for write performance improvement on nand flash memory storage systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 334–337, 2016.

[28] 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. New York, NY, USA: Association for Computing Machinery, 2019, p. 211–223. [Online]. Available: https://doi.org/10.1145/3352460.3358311

[29] C.-W. Tsao, Y.-H. Chang, M.-C. Yang, and P.-C. Huang, "Efficient victim block selection for flash storage devices," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3444–3460, 2015.

[30] S. Wang, F. Wu, C. Yang, J. Zhou, C. Xie, and J. Wan, "Was: Wear aware superblock management for prolonging ssd lifetime," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317929

[31] Y. Wang, L. Dong, and R. Mao, "P-alloc: Process-variation tolerant reliability management for 3d charge-trapping flash memory," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, sep 2017. [Online]. Available: https://doi.org/10.1145/3126554

[32] Y. Wang, J. Huang, J. Chen, and R. Mao, "Pvsensing: A process-variation-aware space allocation strategy for 3d nand flash memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1302–1315, 2022.

[33] J.-N. Yen, Y.-C. Hsieh, C.-Y. Chen, T.-Y. Chen, C.-L. Yang, H.-Y. Cheng, and Y. Luo, "Efficient bad block management with cluster similarity," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 503–513.

[34] J.-G. Yun, G. Kim, J.-E. Lee, Y. Kim, W. B. Shim, J.-H. Lee, H. Shin, J. D. Lee, and B.-G. Park, "Single-crystalline si stacked array (star) nand flash memory," *IEEE Transactions on Electron Devices*, vol. 58, no. 4, pp. 1006–1014, 2011.

[35] W. Zhang, Z. Dong, and Y. Zhu, "Eddysuperblock: Improving nand flash efficiency and lifetime by endurance-driven dynamic superblock management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 1, pp. 95–107, 2022.

[36] Y. Zhou, F. Wu, W. Huang, and C. Xie, "Livessd: A low-interference raid scheme for hardware virtualized ssds," *IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1354–1366, 2021.