SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs

Chun-Yi Liu
Pennsylvania State University
cql5513@cse.psu.edu

Jagadish B. Kotra
AMD Research
Jagadish.Kotra@amd.com

Myoungsoo Jung
Korea Advanced Institute of Science and Technology, KAIST
mj@camelab.org

Mahmut T. Kandemir
Pennsylvania State University
mtk2@cse.psu.edu

Chita R. Das
Pennsylvania State University
das@cse.psu.edu

Abstract

NAND-based solid-state disks (SSDs) are known for their superior random read/write performance due to the high degrees of multi-chip parallelism they exhibit. Currently, as the chip density increases dramatically, fewer 3D NAND chips are needed to build an SSD compared to the previous generation chips. As a result, SSDs can be made more compact. However, this decrease in the number of chips also results in reduced overall throughput, and prevents 3D NAND high density SSDs from being widely-adopted. We analyzed 600 storage workloads, and our analysis revealed that the small read operations suffer significant performance degradation due to reduced chip-level parallelism in newer 3D NAND SSDs. The main question is whether some of the inter-chip parallelism lost in these new SSDs (due to the reduced chip count) can be won back by enhancing intra-chip parallelism. Motivated by this question, we propose a novel SOML (Single-Operation-Multiple-Location) read operation, which can perform several small intra-chip read operations to different locations simultaneously, so that multiple requests can be serviced in parallel, thereby mitigating the parallelism-related bottlenecks. A corresponding SOML read scheduling algorithm is also proposed to fully utilize the SOML read. Our experimental results with various storage workloads indicate that, the SOML read-based SSD with 8 chips can outperform the baseline SSD with 16 chips.

CCS Concepts • **Hardware → External storage**.

Keywords SSD, 3D NAND, parallelism, request scheduling

ASPLOS '19, April 13–17, 2019, Providence, RI
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6240-5/19/04...$15.00
https://doi.org/10.1145/3297858.3304035

ACM Reference Format:
Chun-Yi Liu, Jagadish B. Kotra, Myoungsoo Jung, Mahmut T. Kandemir, and Chita R. Das. 2019. SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs. In *2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), April 13–17, 2019, Providence, RI, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3297858.3304035

1 Introduction

Solid-state disks (SSDs) are an industry preferred storage media due to their much better random read/write performance compared to hard disks. However, as the NAND cell technology node becomes smaller, 2D NAND-based SSDs encounter severe performance and reliability issues, limiting the rate at which the overall SSD capacity increases. The NAND manufacturers addressed this capacity scaling problem by stacking the layers of NAND cells vertically in a 3D fashion.

3D NAND chips [16, 19–21, 28, 33, 45] achieve much higher density by stacking 32, 64, or even 96 layers of cells with an acceptable reliability. For example, the capacity of a 64-layer 3D NAND chip can be as high as 512Gb, requiring only 8 chips to build a 512 GB capacity SSD. As a result, 3D NAND-based SSDs employ fewer chips compared to their 2D-based counterparts. However, such 3D NAND SSDs with fewer chips suffer from *reduced chip-level parallelism*, i.e., reduced number of requests that can be processed in parallel at a given period of time. Unfortunately, this decreased chip-level parallelism can cause severe degradation in performance.

To demonstrate this performance degradation, we compared the IOPS between two generations of 3D NAND SSDs: (1) a Samsung 950 pro SSD [35] using 16 32GB chips and (2) a newer generation Samsung 960 pro SSD [36] using 8 64GB chips. Figure 1 shows the performance comparison between these two SSDs under 4 well-known SSD micro-benchmarks.[1] The higher density SSD fails to outperform the previous generation 950 pro SSD in three read-intensive

[1]These benchmark results are from benchmarkreviews.com, but the similar results can be found in other performance benchmark websites as well.

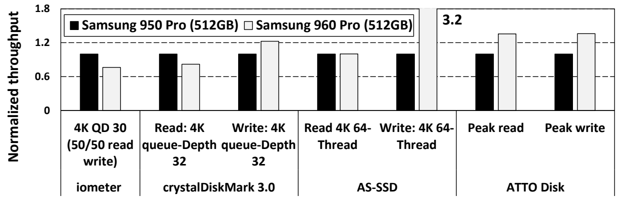


Figure 1. Normalized SSDs' throughput of varying densities.

benchmarks (Iometer, CrystalDiskMark, and AS-SSD), due to the lower multi-chip parallelism.[2] Clearly, this degradation in the overall throughput can prevent 3D NAND SSDs from being widely adopted in the read-intensive applications, such as large graph processing [13, 22–25, 30, 39, 40, 47, 50] and Memcached on SSD [1]. Note also that the write performance is *not* affected by the reduced chip-level parallelism. This is because the write requests are often buffered in a DRAM in SSD and are generally not critical for performance.

To further quantify the degradation on read performance, we analyzed 600 workloads from a repository [27], employing a high-density SSD similar to a newer generation Samsung 960 pro SSD. Our analysis reveals that the sizes (4KB or 8KB) of a majority of read requests are smaller than that (16KB) of a read operation. This disparity between granularities results in low intra-chip read parallelism, where multiple requests wait to be serviced, while an ongoing small request exclusively uses all chip resources. Hence, we rethought the reason behind providing a large read operation granularity in NAND flash, and realized that such large granularity can highly improve the density of 2D NAND. However, this relationship between operation granularity and cell density does *not* exist in 3D NAND, due to the fundamental differences between 2D and 3D NAND micro-architectures.

Motivated by the results and observations above, we propose a novel SOML (**S**ingle-**O**peration-**M**ultiple-**L**ocation) read operation, which can *read multiple partial-pages from different blocks at the same time*, and improve the intra-chip read parallelism significantly. To the best of our knowledge, this is the first work that investigates finer granular read operations for 3D NAND, starting from the circuit-level to evaluating its architectural ramifications. Our main **contributions** in this work can be summarized as follows:

• We analyzed 600 storage workloads to quantitatively show that the read operation is severely degraded due to low multi-chip parallelism in high-density newer generation SSDs. More specifically, most workloads issue small granular reads, which could potentially be executed in parallel in older-generation lower-density SSDs with more number of chips.

• Observing that improving intra-chip parallelism can mitigate the negative impact of the reduced inter-chip parallelism in newer 3D NAND SSDs, we explain the need for reducing the read granularities, and why it is feasible to do so in 3D NAND flash, as opposed to 2D NAND flash.

• Motivated by our observations, we propose to employ finer and more versatile SOML read operation granularities for 3D NAND flash, which can process multiple small read requests simultaneously to improve intra-chip parallelism.

• Building on the SOML read operations, we further propose a read scheduling algorithm, which can coalesce multiple small read operations into one SOML read operation. From our evaluations on an 8-chip high-density SSD, we observed that the proposed SOML read operation and the corresponding algorithm improve throughput by 2.8x, which is actually better than that of a 16 chip SSD of the same capacity. We also compare our approach with three state-of-the-art schemes, and the collected results indicate that our approach outperforms all three.

2 Background and Motivation

2.1 Background

2.1.1 SSD Overview:

A NAND-based SSD (shown in Figure 2) is composed of three main components: (a) an SSD controller, (b) a DRAM buffer [38], and (c) multiple NAND chips. The SSD controller executes flash-translation-layer (FTL) [14, 17], which receives the read/write requests and issues the read/program (write)/erase operations to the NAND chips. The DRAM buffer in an SSD stores meta-data of the FTL and temporarily accommodates the data being written into the NAND chips. The NAND chips are organized into multiple independently-accessed channels. There can be 4, 8 or more channels in an SSD. Each channel contains several flash chips (packages), where they compete for the channel to transfer read/written data. Each chip further consists of 2D NAND or 3D NAND dies, which comply to the same NAND flash standard, ONFI [2]. *Although the exposed interface of 2D and 3D NAND are the same, their micro-architectures are quite different.* For both the NAND types, one die consists of multiple planes, and each plane contains thousands of blocks. Each block hosts hundreds or thousands of pages, depending on the density of the block. Note that a page is the unit for read/write operation, whereas an erase operation is performed at a block granularity. The main difference between 2D NAND and 3D NAND is the block-page architecture, which can be observed in Figure 2. Specifically, the 2D NAND blocks are squeezed side by side on a plane, and the pages in a block are serially-connected. On the contrary, blocks in 3D NAND consist of several vertical slices, whose organization is similar to a 2D NAND block. In 3D NAND SSD, multiple slices share the same set of thick cross-layer signals to overcome the difficulty of transferring high voltage throughput this cross-layer signals. This block-page architecture in 3D NAND introduces both performance and lifetime issues [32, 48].

[2]This relationship between chip parallelism and throughput will be established later through our simulation based study. Also, in some benchmarks, such as sequential read/write, the next generation SSD outperforms the previous generation SSD, as expected.

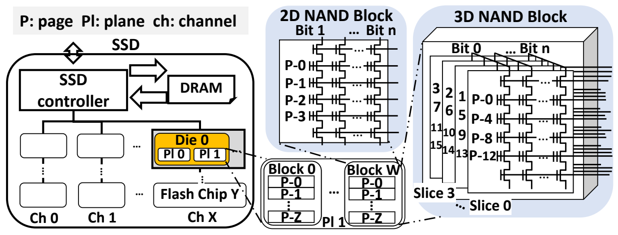


Figure 2. SSD overview and 2D/3D NAND block Organization.

2.1.2 NAND Read Operation:

Due to the short-latency of a read operation, the SSDs are widely used for read-dominant applications, such as large graph processing [50], crypto-currencies [3], and machine learning [11]. As a result, understanding the variations of the read operations is crucial to arrive at techniques that can mitigate the read performance degradation in 3D NAND. Figures 4a and 4b show the granularities and access latencies of various read operations, respectively. As depicted in Figure 4, four types of read operations include: (1) baseline read operation, (2) multi-plane read operation, (3) cache read operation, and (4) random-output read operation. The latency of the (baseline) read operation contains both (1) chip read latency and (2) channel transfer latency. The chip latency is the time taken to read data from the cells (in pages) to the chip internal buffer, while the channel transfer latency is the time elapsed in transferring data from chips' internal buffer to error correction coding (ECC) engines or DRAM buffers in the SSD. Secondly, a multi-plane read operation improves the read throughput by reading the pages in multiple planes at the same time, thereby resulting in a high chip throughput. Thirdly, a cache read operation hides the channel transfer time by employing two sets of internal buffers where one is used for cell data reading and the other is used for data transfer; hence, two operations can be performed in parallel. Lastly, a random-output read operation is used to reduce the data transfer time by only transferring the data required by the requests, *not* an entire page, thereby causing less traffic on the channels.

Although 3D NAND chips support the advanced read operations, the read performance of a single chip still degrades substantially as the chip density increases. This is because, as the chip density increases, a longer cell sensing time (chip read latency) is needed to read the cell data. In fact, the read latency can be as high as $90\mu s$ [44]. In contrast, the channel transfer latency can be highly shortened by a faster data clock-rate, but this enhancement can only slightly reduce the overall read latency. Therefore, the chip read latency is the *bottleneck* for the read performance.

2.1.3 Additional Read Operations by ECC:

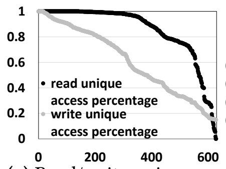
Due to high-density and advanced multi-bit-cell technology, the reliability of 3D NAND cells is an important issue; hence, strong error-correction-coding (ECC) techniques, such as low-density-parity-checking-code (LDPC), are required to guarantee data integrity. Modern ECCs use the parity information in the page and the additional sensing (re-read) operations to correct data. Such re-read operation employs a read-retry operation to re-adjust the chip sensing voltage, so that different values can be read from the page. Then, using the original and re-read values, the ECC can correct more erroneous bits, which can extend the overall SSD lifetime by up to 50% [4]. Those additional read operations prolong the latency of the read requests, degrading the read performance.

2.2 Motivation: Workload Analysis

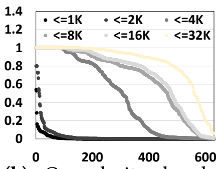
As explained in Section 1, the read performance of higher density (lower multi-chip parallelism) SSDs is *not* good for several well-known benchmarks. To understand the performance degradation better and address it effectively, we used the SSDSim [15] simulator and evaluated over 600 storage workloads from OpenStr [27]. The behaviors of the various evaluated workloads are plotted in Figures 3a and 3b, and the parameters of the simulated SSDs can be found in Table 1. In these experiments, we used three *iso-capacity* SSD configurations, denoted by (number of chips, capacity per chip), namely: (A) (32, 16GB), (B) (16, 32GB), and (C) (8, 64GB). The detailed parameters of the various density chips evaluated are taken from papers [16, 19, 20]. Note that, to have a fair comparison, the number of channels across these three configurations is set to 4, so that the configurations have the same channel parallelism. Note also that we use the same read/write latency to illustrate the low multi-chip parallelism issue.

Figure 3c shows the read/write throughput of our workloads, which are sorted by throughput[3]. As the number of chips decreases, the performance drops significantly. On average, SSD (C) is about 1.5x slower than SSD (A). This is because, in SSD (C), fewer chips can process the read/write requests simultaneously, resulting in reduced chip-level parallelism. To acquire more specific information, the average read and write latencies of the workloads are presented in Figure 3d. It can be observed from this figure that, the write latencies of most workloads are really low compared to their read latencies, as the number of chips is reduced from 32 to 8. This is due to the presence of a large DRAM buffer in SSD. More specifically, the DRAM buffer, which is set to 128MB in our simulation, can temporarily cache the write requests, and drain them to NAND chips later during the idle periods. Therefore, the latency of write requests can be as short as the DRAM access latency, provided that the DRAM buffer is large enough. One should note that the current 1TB 3D NAND SSDs are equipped with more than 1GB DRAM buffer [36]; however, we used 128MB of DRAM buffer, conservatively.

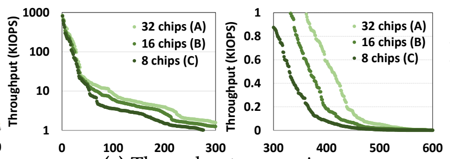
[3]Note that the figures in this section are sorted by the corresponding metrics, to clearly show the trends across workloads.



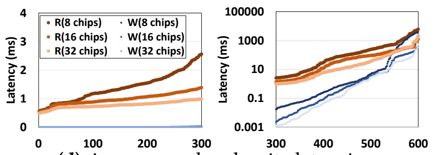
(a) Read/write unique access.



(b) Granularity breakdown of read requests.

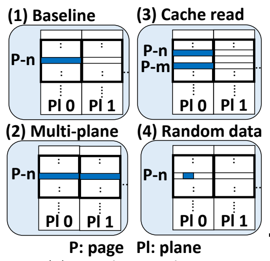


(c) Throughput comparison.

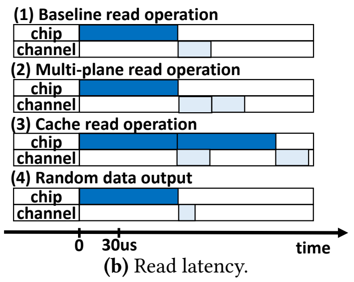


(d) Average read and write latencies.

Figure 3. Results of our analyses of over 600 workloads (the x-axes represents individual traces sorted in an increasing/decreasing order).



(a) Read granularity.



(b) Read latency.

Figure 4. Overview of read operations.

On the other hand, as can be observed from Figure 3d, the average read latency increases as the number of chips decreases, due to increased contention for the chips. Different from the write requests, the DRAM can only provide a limited read performance improvement. This is because the DRAM-induced read improvement comes primarily from the cached recently-read data, which is rarely re-read in a short time. Figure 3a plots the fraction of addresses that are only read once for each workload. As can be observed from the figure, over 80% of the addresses are read only once in the first 400 workloads; so, the DRAM *cannot* reduce the read latency by caching the read data. The main reason behind such poor temporal locality for reads is that the frequently-read data are typically cached in the upper layers.

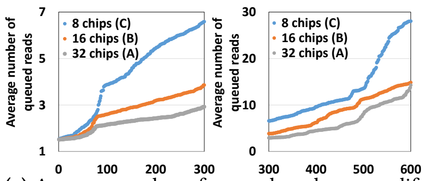
To further understand and improve the performance of read requests, we analyzed the following two metrics: (1) the average number of read requests queued per chip (Figure 5a), and (2) the granularity of the read requests itself (Figure 3b). Figure 5a shows the queued read requests for 3 systems under 600 different-strength trace-based workloads. In the figure, 3 systems may queue different numbers of read requests based on the system throughput. For example, due to its higher throughput, the 32-chip system can keep the queue length lower than 5, while the 8-chip system experiences longer request queues. *To compensate for the parallelism lost due to the reduced number of chips, the intra-chip parallelism needs to be improved.* Figure 3b shows the possibility of processing multiple read requests at a time by breaking down the read request granularity across the workloads. Note that the current 3D NAND pages are at least 16KB, which is the *minimum unit (granularity)* for a read operation. As can be seen from this figure, there are only few workloads dominated by 1K or 2K read requests. However, plenty of workloads are dominated by 4K and 8K read requests. Specifically, over 300 and 480 workloads mainly contain read requests that are smaller than 4K and 8K, respectively. Only a handful of workloads warrant large granularities, such as 16K and 32K. To that end, the large amount of smaller read requests, viz. 4K and 8K, can be serviced by the larger page sizes, which can be 16K or 32K.

To summarize, the performance degradation caused by the fewer number of chips only impacts the read requests, not the writes. Also, a majority of the workloads issue read requests which are much smaller than the 16KB reads.

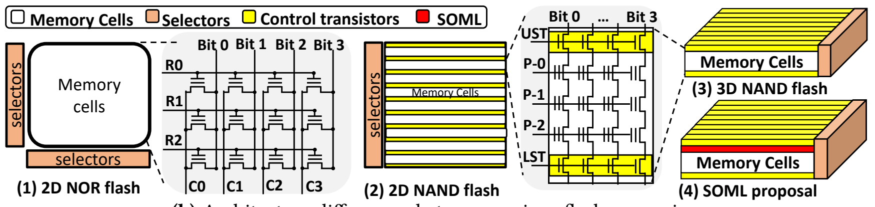
3 Overview

The read-dominated workload characteristics, as presented in Section 2.2, clearly indicate that the read request granularity is much smaller than the size of a baseline page read operation supported in 3D NAND. This observation motivated us to rethink the granularity of basic read operation in NAND flash. We want to emphasize that, the page-granular read operation in 3D NAND is inherited from 2D NAND. As shown in Figure 5b-(2), the page-granular read operation in 2D NAND stems from the uni-direction selector, which can only select one page at a time. On the contrary, in NOR flash (shown in Figure 5b-(1)), a bi-directional selector can be used to index and access a single data cell. However, such fine-granular operations require larger space between NOR data cells; as a result, the cell density of NOR flash is sparser than that of NAND flash. Consequently, current high dense SSDs only use NAND flash as their storage media.

Enabling fine-granular read operations via bi-directional selectors (like 2D NOR flash) in 2D NAND flash is *infeasible.* This is because, additional selectors need to be practically added between all the blocks in 2D NAND to enable bi-directional selector; but, doing so would reduce the overall cell density significantly. In contrast, 3D NAND, shown in Figure 5b-(3), can enable such fine-granular operation, since it employs a completely different block-page architecture, where the control transistors reside in the top and bottom layers, and not between blocks. Therefore, by inserting additional selector layers (depicted in Figure 5b-(4)) between the top and data-cell layers, we can enable the fine-granular



(a) Average number of queued reads across different workloads.



(b) Architecture differences between various flash memories.

Figure 5

read operation in 3D NAND, *without reducing the cell density of 3D NAND.*

In this paper, we propose "*single-operation-multiple-locations (SOML)*" read operation to boost the intra-chip read performance *without* reducing the NAND chip's storage density. The basic idea behind our SOML read operation is to execute multiple small read requests simultaneously with a single SOML read operation. The latency of a SOML read is slightly higher than that of the baseline read operation. Our SOML read operation, which encompasses small read requests, imposes two constraints. The smaller read requests should share (1) the same bit-lines and (2) block-decoders. The density of highly-condensed bit-lines *cannot* be doubled to enable multiple concurrent read accesses; as a result, the read requests in our SOML read operation must share the existing bit-lines. On the other hand, the block-decoders are shared across different blocks, even with the duplicated block-decoders. Besides those two constraints, our SOML read operation warrants additional control layers and peripheral circuits to be added to the 3D NAND chip. The hardware changes required by our SOML read operation are discussed in Section 4. Figure 6a depicts an example working of our SOML read operation, compared to the baseline read. The baseline read can only read a whole page (16KB) in a block at a time, agnostic to the read-request granularity. In contrast, our SOML read can read two half pages at the same time, where the first half page-1 of block-0 and second half page-0 of block-M are read together. In this example, read performance is nearly doubled, provided that these two requests only need the data in the read half pages.

To fully utilize our SOML read operation, the SSD management software (FTL) needs to further include an algorithm to find and combine the multiple small read requests into a single SOML read operation. Clearly, this algorithm should be able to select the multiple small read requests which satisfy the hardware constraints mentioned before.

4 SOML Read: Hardware Modifications

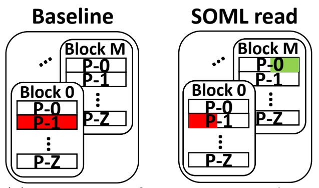
In this section, we discuss the peripheral circuitry modifications, overheads involved, and command formats of our proposed SOML read operation. We also cover the alternate design options along with other concerns.

4.1 Peripheral Circuit Modifications

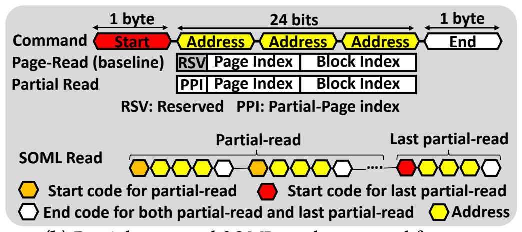
To enable SOML read, 3D NAND has to support the following two mechanisms: (1) partial-page read operation and (2) simultaneous multiple partial-read operations across blocks.

Partial-page read operation: A partial-page read operation reads only part of a page, typically *half* or *quarter* of a page, and transfers the read data to the NAND internal buffer via the bit-lines. Before describing our circuit changes, let us discuss the baseline read operation in 3D NAND (shown in Figure 6c). Only one page can be read across all the blocks in a plane. While reading a page in a block, the chip un-selects all the other blocks via the block-decoder (BD) (shown in Figure 6c-(2)), so that only one block receives the control signals from corresponding page-decoder (PD). The page decoder indexes the read page by a layer signal and a corresponding slice signal. Note that the other layer and slice signals would be set to appropriate values to indicate "Off," so that only the corresponding page is read. Figure 6c-(3) shows how to correctly set the control signals to read the data in one of the cells in page-4, which resides in slice-0 of block-0 in Figure 6c-(1). The voltage of page-4 (layer-2) signal is set to V_{read}, while the voltages of page-0 (layer-1) and page-8 (layer-3) are set to V_{pass}, to ensure that only the value stored in page-4 is drained out via the bit-lines to the sensing circuits. The lower select transistor (LST) is set to V_{cc} in the case of read operation, while the upper select transistor (UST) is used as the slice signal in 3D NAND. More specifically, the UST of the selected slice is set to V_{cc}, while the USTs of other slices in the same block are set to 0V. Therefore, although other slices receive the same set of layer signals, the data will *not* drain out and interfere with the read operation, thereby inhibiting reading from other slices, as shown in Figure 6c-(4).

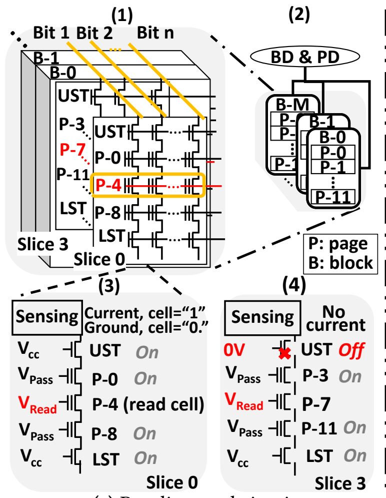
In our SOML read, to enable the partial-read operation, all pillars (shown in Figure 6d-(3)) need to accommodate additional SOML select transistors (SOML ST), which reside between the UST and the first cell. That is, the additional SOML ST layers are inserted between the UST layer and the first cell layer. The usage of SOML STs can be found in Figure 6d-(1), where the first-/second-half pages of SOML STs across multiple slices in a block can be selected by different control signals (the red lines). Hence, adding these two additional signals to the page decoder, we can enable



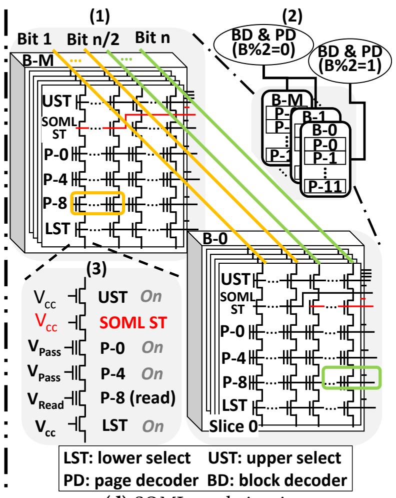
(a) Overview of our SOML read operation.



(b) Partial-page and SOML read command format.



(c) Baseline read circuit.



(d) SOML read circuit.

Figure 6

the half-page read operation. Note that a finer partial-page read operation can be achieved by adding more SOML STs and the corresponding signals.

Simultaneous multiple partial-read operations: Simultaneous multiple read operations *cannot* be performed by the peripheral circuits in the baseline 3D-NAND architecture due to the shared control circuitry among read operations. To enable such multiple read operations, the shared circuitry, namely, the page-decoder (PD) and block-decoder (BD), have to be replicated or modified. Hence, the pages from different blocks can be accessed by different sets of page-decoders and block-decoders. Figure 6d-(2) shows the modified circuitry to perform two half-page read operations. The page decoder is duplicated to index two distinct half-page read operations, while the block decoder is split into two smaller block decoders, where each of them can index a half set of blocks. Note that the number of bit-lines remains the same, since doubling the highly-condensed bit-lines is *not* practical. Note also that, we can only enable multiple read operations across different blocks; multiple read operations in the same block *cannot* be parallelized due to the 3D NAND block-page micro-architecture. In summary, with the SOML select transistor layers and additional block-/page-decoders, our proposed SOML read operation can be realized.

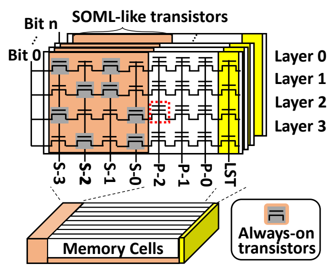


Figure 7. The 3D VGNAND.

Hardware Feasibility: Our proposed circuitry is an extension of an early 3D horizontal-channel (vertical-gate) NAND, called VGNAND [21]. Note that, as opposed to VG-NAND, the current mainstream 3D NAND flash (shown in Figure 5b-3) is vertical-channel based. In VGNAND, to access a page, multiple pages on the same corresponding location across layers are read out, and the SOML-like transistors select only one page among them to access. For example, in Figure 7, to read the data in Page-2 on Layer-2, the signal P-2 is set to V_{read} (while P-0 and P-1 are set to V_{pass}), and only S-1 and S-3 are set to On (while S-0 and S-2 are set to Off). Our proposed circuitry is similar to the VGNAND, but we stack the select transistors on the top and use them to select partial-pages across different blocks. Note that, to clearly demonstrate the effect of SOML read in Figure 6d, we abstract the details of SOML transistors. (that is, in reality, 4 layers of SOML transistors are needed to enable a quarter partial-page read).

4.2 Hardware Overheads

The hardware overheads of SOML read operation come mainly from the SOML transistors and decoders. The SOML transistors account for the major transistor overheads; however, the area and storage density of the chip is *not* affected, since we only increase the number of transistor layers. Note also that, although the density of 3D NAND is achieved by layering more 3D NAND cells, the difficulty in achieving more layers is in the mechanism needed to squeeze more data cell layers in a limited distance between the top and bottom data cell layers. On the other hand, the additional decoders only introduce fewer additional transistors, but the area of the chip increases due to the block-decoders. Although we only split the baseline block-decoder into multiple smaller block-decoders, the area of the overall block-decoders still increases. This is because, the area of a modern highly-optimized decoder is *not* linearly proportional to the number of indexable blocks; so, we estimate that the additional block-decoders would yield 3x area overhead over the baseline if a quarter page SOML read operation is enabled.

The area overhead of the proposed quarter page SOML read operations can be calculated as follows: The peripheral circuits in 3D NAND flash chip constitute 7 ~ 9% [16, 19, 20], and the block-decoder and page-decoder occupy about 7% and 4% of the peripheral circuits [34], respectively. Therefore, the overall hardware overhead is at most 1.7% of the whole area, which indirectly reduces density by 1.6%.

4.3 SOML Read Command Format

Current read-related commands cannot be used to issue our proposed SOML read operation, since they do *not* support the following two SOML read mechanisms: (1) indexing the partial-page and (2) issuing multiple partial-page across multiple blocks. Therefore, to issue a SOML read operation, we introduce two new commands: (1) partial-page read command and (2) SOML read command.

The partial-page read command, which is shown in Figure 6b, is modified from the baseline read operation. The only difference is that few bits in the reserved address field are used for indexing the partial-page in a page. On the other hand, the SOML read command consists of a sequence of partial-read commands. To notify the chip on how many partial-read operations are combined into one SOML read, we introduce another variation of the partial-read command, namely, *last partial-read command*. This new command notifies chips the last partial-read command out of a sequence of partial-read commands via a different start code (shown in Figure 6b). Thus, all previously-issued partial-read commands and the last partial-read command are combined into one SOML read. It is to be noted that such SOML read command sequence is practical, since the existing multi-plane read operations are essentially issued in the same manner. The command overhead of the SOML read operation is in nano-second range, which is negligible compared to the hundred μs read latency. Note also that it is FTL's responsibility (covered in Section 5), to guarantee that a sequence of partial-read (wait) command(s) do *not* violate any constraint of our SOML read operation, such as shared bit-lines and decoders.

4.4 Discussion of the SOML Read Operation

4.4.1 The asymmetric latency of partial-read operation:

Due to the high-density NAND flash demand, 3D NAND manufacturers ship MLC (multi-level cell) and TLC (triple-level cell) flash, where one cell can store 2 and 3 bits of information, respectively. The read latencies of different bits in a cell are different. That is, the second bit in a cell requires additional read sensing operations compared to the first bit in the same cell, thereby increasing the read latency of the second bit. Such asymmetric read latencies in 3D NAND may result in sub-optimal read performance, since a SOML read can only start transferring out data from the chip internal buffer to the DRAM buffer after all partial-reads have been successfully performed. As a result, short-latency partial-reads need to wait for the completion of long-latency partial-reads, which prolongs the latency of the short-latency read requests.

4.4.2 Read disturbance:

Read disturbance [7, 29] becomes a major reliability concern in the high-density NAND flash. This is because, as the number of pages per block increases, more pages in a block are subjected to the disturbance from the page read operation. Such accumulated read disturbance will ultimately alter the value stored in the NAND cells, even with a strong ECC protection. Our proposed SOML read operation does *not* worsen the read disturbance, since the partial-page read operations in a SOML read operation read the partial-page in distinct blocks; as a result, the read disturbances in different blocks do *not* deteriorate each other.

5 SOML Read: Software Modifications

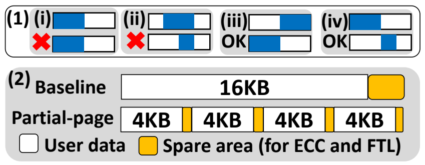
The SOML read operation presented in Section 4 requires software changes in identifying the read requests that satisfy the hardware constraints of the SOML read operation. To that end, we propose a novel *scheduling algorithm* in the FTL layer to construct a SOML read operation. Since our algorithm schedules the FTL-translated (physical-address) read requests, it is easily applicable across different FTL implementations.

5.1 SOML Read Operation Constraints

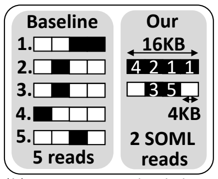
The hardware and software constraints in constructing a SOML read from partial-reads include: (a) a shared bit-line across all partial-reads and (b) a shared block-decoder between the blocks corresponding to the partial-reads.

Shared bit-lines: Due to the highly-condensed bit-lines, to form a SOML read operation, multiple partial-read operations have to share the same set of bit-lines. Figure 8a-(1) shows four different scenarios for two partial-read operations. The first and second pairs of the partial-read operations *cannot* be executed simultaneously, since these partial-reads compete for the same set of bit-lines to transfer the read data to the chip-internal buffer. In contrast, the third and fourth pairs can be executed at the same time.

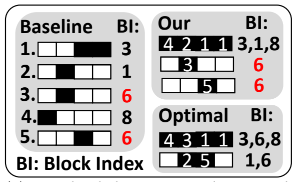
We assume that the data layout of a page is modified as shown in Figure 8a-(2), to accommodate the partial-read operation. Specifically, instead of dividing the page into only two mandatory regions, i.e., user data region and spare area region in the baseline layout, we propose splitting the spare area region into smaller regions, thereby associating each partial-page with a corresponding spare area. As a result, the partial-page and its corresponding spare area are now contiguous. Note that splitting spare area region does not harm the ECC capability. This is because the ECC in modern SSDs does not use the entire data region (16KB) as a unit to



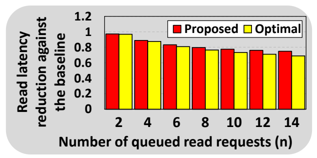
(a) Examples and page layouts of shared bit-lines constraint in (1) and (2), respectively.



(b) A scheduling example considering the shared bit-lines constraint.



(c) A scheduling example considering both the shared bit-lines and block-decoders constraints.



(d) Comparison between our proposed algorithm and the optimal algorithm (lower is better).

Figure 8

```
Algorithm 1: SOML scheduling algorithm
  Input: R_queue: queued read requests
  Data: used_BLs: bit-lines, used_BDs: block-decoders
  SOML_reqs ← $\phi$; used_BLs ← $\phi$; used_BDs ← $\phi$;
  for req in R_queue do
      if BLs-overlapped(req.BLs, used_BLs) then continue ;
      if BDs-overlapped(req.BDs, used_BDs) then continue ;
      used_BLs.insert(req.BLs);
      used_BDs.insert(req.BDs);
      SOML_reqs.insert(req);
  return SOML_reqs;
```

encode. Instead, the data region is broken into small chunk (1KB or 2KB) and each chunk is encoded separately.

Shared block decoder: To enable executing multiple partial-read operations simultaneously, we split the baseline block-decoder into smaller block-decoders. Each smaller block-decoder can only access a disjointed sub-set of blocks, which *can* only execute one partial-read operation. Hence, to perform a SOML read operation, the contained partial-read operations have to be executed by different block-decoders. For example, in Figure 6c-(2), one block-decoder is split into two; as a result, a partial-read can be performed on odd blocks, while another one can only be performed on even blocks. More specifically, block-0 and block-M (where M is an odd number) can execute two partial-read operations individually. However, block-1 and block-M *cannot*, since they are controlled by the *same* block-decoder.

5.2 Scheduling Algorithm

Algorithm 1 gives our proposed SOML read scheduling algorithm. This algorithm considers both (a) shared bit-lines (BLs) and (b) shared block-decoders (BDs), while combining partial-reads to form a SOML read operation. Our greedy algorithm iterates over all the read requests and combines reads that satisfy the imposed hardware constraints.[4]

Figure 8b shows how Algorithm 1 finds the best set of SOML read operations. As can be observed, there are 5 pending read requests queued to be serviced by a chip. Let us assume that their granularity is either 4KB or 8KB, which are the sizes of a quarter- or half-page, respectively. The first round of Algorithm 1 picks requests 1, 2 and 4 to combine them into one SOML read operation. This is because, requests 3 and 5 share the bit-lines with requests 2 and 1, respectively; as a result, requests 3 and 5 *cannot* be combined to be part of the same (first) SOML read operation, resulting in an additional SOML read operation, as depicted in Figure 8b. Note that this set of SOML read operations is optimal in this example, since the total size of read requests (24KB) warrants at least two read operations (16KB).

However, Algorithm 1 can occasionally generate sub-optimal sets of SOML reads, since the SOML read has the mentioned two constraints, and NAND flash has asymmetric read latencies. Figure 8c shows a sub-optimal example caused by the shared block-decoders. In this example, the block indices of the read pages are indicated as *BI*; so, both the imposed hardware constraints have to be considered. In this case, the first SOML read is the same as the one shown in Figure 8b, since requests 1, 2, and 4 do *not* share any block-decoders. However, since requests 3 and 5 read the same block and hence share the same block-decoder, they *cannot* be performed simultaneously. Such a scenario can be optimized by combining requests 1, 3, and 4 in the first SOML read, and then requests 2 and 5 can form the second SOML read.

Such sub-optimal case can be handled by employing an "optimal algorithm," which considers all SOML read operations simultaneously; however, such an "optimal algorithm" incurs an exponential time-complexity, making it impractical[5]. Hence, to see whether the SOML read operations need to be scheduled by an optimal or other near-optimal algorithms, we ran experiments to observe the difference between our proposed and optimal algorithms. We used randomly-generated workloads, which contain 4K, 8K, 12K and 16K requests, to cover as many queued-request scenarios as possible; hence, one can realize how rare these sub-optimal examples are encountered. The latencies of the requests can span any of 3 distinct TLC read latencies (shown in Table 1). We randomly generated 7 workloads, and each workload contains 10000 sets of fixed number (2 to 14) of queued read requests.

Figure 8d shows the read latency reductions brought by the proposed and optimal algorithms, compared to the baseline scheduling algorithm. As the number of queued read requests (X-axis) increases, the optimal algorithm gradually

[4]Note that the read request queue is in chronological order, so the head of the queue holds the oldest request.

[5]The "optimal algorithm" spends more than one hour on a desktop CPU, to find the optimal SOML read combination for 14 queued reads. In contrast, the proposed algorithm only have to go over the queue once, which takes less than few microseconds.

performs better than our proposed algorithm. However, this latency reduction difference between our proposed and optimal algorithms is much smaller than that of between the baseline and our proposed algorithm. This means that any additional benefit that could be obtained by implementing a costly (exponential-time-complexity) optimal algorithm would be minimal; and so, we believe that our proposed algorithm is sufficient to schedule and combine the SOML reads.

6 Experimental Evaluation

6.1 Setup

Baseline 64-layer 3D NAND chip parameters [20]	
(Die, Plane, Block, Page)	(1, 2, 1437, 768)
(Page size, Cell density)	(16KB, TLC)
(Program, Erase)	(900μs, 10ms)
TLC read latency (LSB, CSB, MSB)	(90μs, 120μs, 180μs)
Smallest partial-read size	4KB
Chip capacity	64 GB
SOML read enabled 64-layer 3D NAND chip parameters	
Max SOML read	4 partial-reads
TLC read latency (LSB, CSB, MSB)	(92.7μs, 123.7μs, 185.5μs)
SSD parameters	
(number of chips, DRAM capacity)	(8, 128MB)
(FTL, GC trigger)	(Page-level mapping, 5%)
Victim block selection	block with max #invalid pages
Transfer time per byte	5ns
(Over provision, Initial data)	(25%, 50%)
#Re-read operations	3
32GB 3D NAND chip parameters [19]	
(Die, Plane, Block, Page)	(1, 2, 1888, 576)
16GB 3D NAND chip parameters [16]	
(Die, Plane, Block, Page)	(1, 1, 2732, 384)

Table 1. Characteristics of the evaluated SSDs.

trace	read %	unique access	size < 2KB	size < 4KB	size < 8KB	size < 16KB	size < 32KB
24HRS8	70.3	0.29	0	0.03	0.43	0.43	0.44
BS78	55	0.13	0	0.98	0.98	0.99	0.99
casa21	7.1	1	0	0.96	0.97	0.98	0.98
CFS13	63.2	0.78	0	0.84	0.85	0.86	0.94
ch19	14.9	0.76	0	1	1	1	1
DDR20	90.6	0.44	0.19	0.69	0.72	0.74	0.79
Ex64	22.3	0.96	0	0.13	0.79	0.83	0.87
hm_1	93.8	0.02	0	0.01	0.87	0.87	0.88
ikki18	1.1	1	0	0.83	0.9	0.98	0.98
LMBE2	82.9	0.88	0.01	0.14	0.16	0.21	0.35
mds_0	98.2	0.89	0.05	0.53	0.56	0.61	0.68
prn_1	85.4	0.41	0	0.44	0.64	0.68	0.71
prxy_0	5.66	0.07	0.01	0.83	0.86	0.89	0.96
src1_2	16.6	0.18	0.02	0.55	0.62	0.68	0.77
src2_0	12.7	0.29	0.02	0.82	0.85	0.89	0.98
stg_1	93.0	1	0.02	0.05	0.06	0.06	0.07
web_1	85.4	0.96	0.11	0.23	0.25	0.27	0.31
w8	15.4	1	0	0.96	0.96	0.98	0.99

Table 2. Important read characteristics of our workloads. Columns 4-8 give the read request granularity breakdown.

We use SSDSim [15] to quantify how much SOML read operations can improve the 3D NAND intra-chip read parallelism. The detailed SSD parameters used in our evaluations can be found in Table 1. We simulated a 512GB capacity SSD, whose configuration parameters are very similar to commercial SSDs such as [36]. We used 18 workloads[6] from the OpenStor [27] repository. The details of these workloads are given in Table 2.

Due to the additional SOML transistors, the latencies of all read-related operations are prolonged. The increased read latency can be calculated by the following two equations [34]:

$$Read\ Latency = \frac{C}{I}\triangle V, \quad \text{and} \quad I = \frac{V_{BL}}{(N-1)R}, \qquad (1)$$

where C, $\triangle V$, and V_{BL} are the capacitor capacitance, measured voltage, and bit-line-applied voltage used by the sensing circuits, respectively. These three parameters remain the same as in the baseline, since our SOML read does not change the sensing circuits. N is the number of data cells/transistors between the upper and lower select transistors, which is also the number of layers in 3D NAND, and R is the resistance of a cell/transistor. We use a worst case estimation, where the added SOML transistor has the same worst resistance as a data cell. Therefore, the increased read latency can be calculated as $\frac{64+4-1}{64-1} = 1.063$ times of the baseline read latency, if a quarter partial read is enabled.

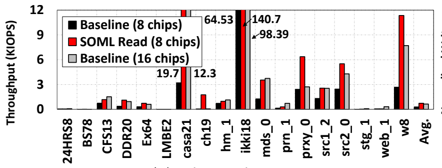
6.2 Results

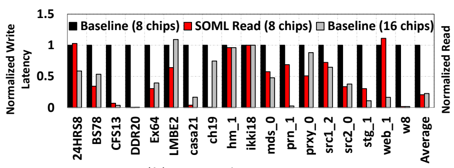
6.2.1 SOML Read Performance

Throughput: To show the intra-chip read parallelism improvement brought by the SOML read operation, we compared the following three systems: **(a)** a baseline SSD with 8 chips, **(b)** a SOML read-enabled SSD with 8 chips, and **(c)** a baseline SSD with 16 32GB chips. Figure 9a plots the read/write throughput comparison between these three systems. On average, our proposal achieves about 2.8x better throughput than the baseline under the same number of chips (8 chips). It can also be observed that, our SOML-enabled system outperforms both of the baseline systems tested, since one SOML read operation can execute up to 4 (4KB) partial-read operations simultaneously. As a result, the read performance can be highly enhanced, thereby improving overall throughput.

Read/write latency: To further understand the reason behind the observed performance improvements, we plot the write and read latencies in Figures 9b and 9c, respectively, normalized to the 8-chip baseline. Although only reads can be executed in parallel by the SOML read, the write latency reduction is also significant, and is even higher than that of reads. This is because, in SSDs, the writes typically have lower priority compared to the reads due to the long-latency of the former. As a result, to get processed, the writes need to wait for the completion of all queued reads. Consequently, shortening the overall read latency via the SOML reads shortens the write latencies as well.

[6]Some of the workloads used are abbreviated as follows: ch19=cheetah19, Exchange64=Ex64, and webusers8=w8.


(a) Throughput.


(b) Write latency.

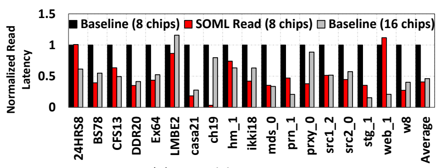
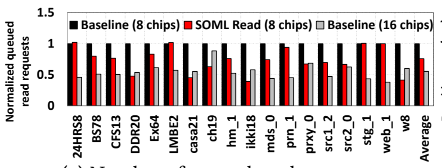
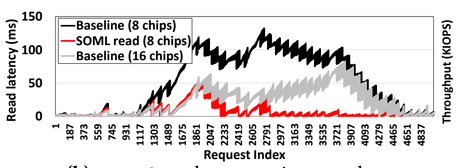

(c) Read latency.

Figure 9. Performance comparisons between the baselines and SOML read.


(a) Number of queued read requests.


(b) prxy_0 read request time graph.

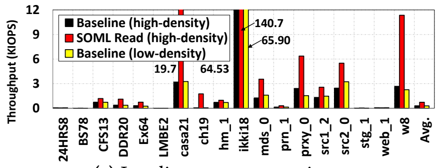
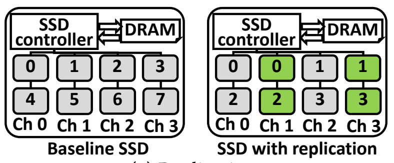

(c) Iso-die-count comparison.

Figure 10. Various comparisons between the baselines and SOML read.


(a) Replication.

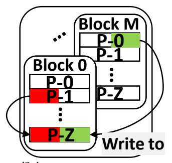

(b) Remapping.

Figure 11. Compared schemes.

Number of queued read requests: Figure 10a shows the average number of read requests queued across all chips for the three systems tested. Our proposal can successfully reduce the number of read requests queued, compared to the baseline system with 8 chips. However, the baseline with 16 chips still outperforms our proposed system. This is because, under the same number of requests, the increased rate of queued reads on an 8-chip SSDs is higher than that of 16-chip SSDs. Therefore, though our SOML read can process multiple queued requests simultaneously, the number of queued reads is still high. Note however that, the throughput of our SOML read system outperforms the baseline system with 16 chips.

Time graph: Figure 10b plots the latency comparisons of the first 5000 read requests of prxy_0 across three systems. As can be observed, the read latencies of our proposed system are much smaller than those of the two baseline systems. This is because, our system utilizes the SOML read operation to simultaneously process multiple read requests; hence, fewer read requests are queued, shortening the incurred read latency.

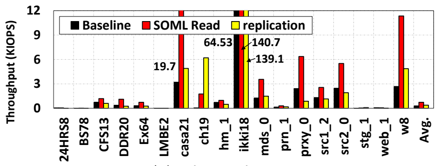
Iso-die-count comparison: [7] Figure 10c gives the performance comparison between the baseline and SOML read with 8 high-density (64GB) and low-density (32GB) chips. As can be seen, SOML read can still highly improve the performance. Nevertheless, only a limited performance difference between baseline high-/low-density systems is observed, since the systems share the same level of inter-chip parallelism under the iso-die-count comparison. One may think that increasing number of NAND chips can solve the SSD lower parallelism issue, but such solution increases the SSD capacity; and increasing the SSD capacity makes the mapping table larger and requires a re-design of SSD processor-DRAM architecture, which presents an undesired overhead for SSD vendors. In comparison, our SOML read improves the SSD parallelism without modifying the SSD architecture.

6.2.2 Comparisons with Other Schemes

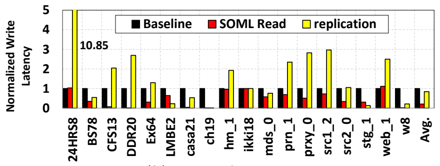
Replication: The main idea behind replication is to keep more than one copy of data in different chips across an SSD. Hence, a read request can be serviced by any one of the chips with a copy of the data. This idea, also known as RAIN (Redundant Array of Independent NANDs) [5], is inherited from RAID (Redundant Array of Independent Disks). Among all the proposed RAIN types, only RAIN 1 based approaches (shown in Figure 11a), which duplicate all data, can improve the read performance. This is because this particular RAIN system can service multiple queued read requests from different chips with replicated data, leveraging the multi-chip parallelism. However, it needs to be emphasized that, RAIN 1 necessitates maintaining *coherency* across replicas. As a result, a write incurred by a replicated data further results in multiple writes to the same data in other chips to maintain coherency, resulting in more number of writes.

Figures 12a, 12b, and 12c plot the throughput, average write and read latencies of the compared schemes, respectively. Note that this is an *iso-chip-count* comparison, since, when using replication, half of chips are used for additional data copies; so, the SSD capacity of the replication is half of the baseline and our system. As can be observed, although, on average, replication can perform better than the baseline, it is still much worse than our proposed SOML read-enabled system. This is because, the replication introduces additional writes to guarantee the data consistency across different chips. Due to such additional writes, the replication performs

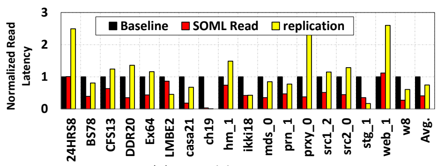
[7] We use the terms "chip" and "die" interchangeably, since, in our experimental setup, each chip has one die.



(a) Throughput.

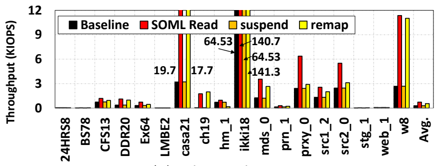


(b) Write latency.

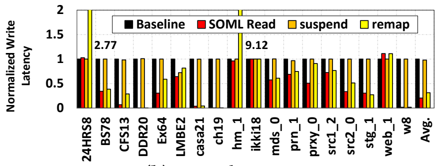


(c) Read latency.

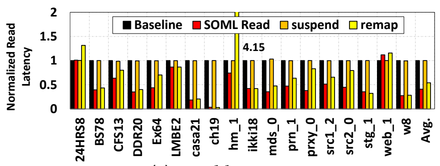
Figure 12. Performance comparisons between SOML read and the replication-based approach.



(a) Throughput.



(b) Write latency.



(c) Read latency.

Figure 13. Performance comparisons among SOML read, the remapping, and the program/erase suspension.

worse in workloads such as prxy_0 and src2_0. Therefore, the replication is *not* a practical option for improving intra-chip read parallelism.

Remapping: Remapping idea, borrowed from a DRAM-based study, Micropages [26, 37], copies the data being accessed simultaneously to the same access-unit, so that it can be accessed faster in a single read operation next time. Figure 11b shows how the remapping technique can be used in reducing the read performance degradation in 3D NAND-based SSDs. For example, the first half of page-1 in block-0 and the second half of page-0 in block-M are always accessed at the same time; as a result, to improve the read performance, the data of two half pages can be copied into another page (page-Z of block-0 in the example). Therefore, two requests can be serviced by only one baseline read operation in future, without any hardware modification.

Figures 13a, 13b, and 13c show, respectively, the throughput, average write, and read latencies of different schemes. On average, our SOML read-enabled scheme outperforms the remapping-based scheme, since the additional write operations, introduced by the remapping technique, degrade the overall performance. However, in some workloads, such as Ent12 and Ch19, the remapping-based scheme performs slightly better than our proposed scheme. This is because, these workloads have easily-predictable, repeatedly-accessed patterns; thus, combining multiple such requests can lead to significant performance improvements. Note however that those read patterns are *not* frequent, as mentioned in Section 2.2 and illustrated in Figure 3a. In summary, our SOML read operation is more general and can improve the intra-chip parallelism in most of the workloads.

Suspension of program and erase operations: The suspension of program and erase operations is proposed in [43]. The reason why suspension can improve the read performance is due to the asymmetric operation latencies exhibited by NAND flash, where the write and erase latencies are more than 10 times longer than the read latency. Hence, a read operation may be blocked by an ongoing write or erase operation. To avoid such undesirable read operation blocking, the write and erase operations can be suspended to allow the blocked reads to be processed. Therefore, the overall read performance is *not* affected by any write or erase operations.

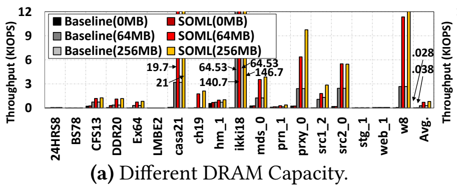
In our implementation of this idea, we assume that *perfect* write and erase suspensions are employed in the NAND chips; as a result, the read operations will *not* be blocked by any write or erase operations. However, due to overheads incurred by the preempted read operations, the latencies of the write and erase operations are prolonged. Clearly, this implementation is too optimistic to be employed by the NAND chips, since the overheads brought by write suspension for 3D NAND would be very high due to the full-sequence-program operation [20].

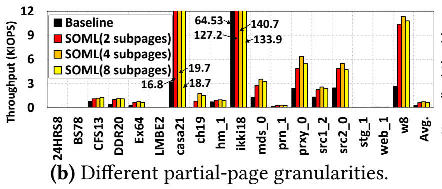
The comparison results can be observed in Figures 13a, 13b, and 13c. Our SOML read-enabled scheme outperforms the program/erase suspension scheme, since the latency difference between read and write operations is shortened, owing to the prolonged chip read latency and the faster full-sequence-program (write) operation. Consequently, fewer reads are blocked by writes. In addition, the number of erase operations incurred during GC is also reduced, due to the reduced number of 3D NAND blocks per plane as the block size increases. Therefore, the suspension technique can only slightly improve the overall performance.

6.2.3 Sensitivity Results

To demonstrate that our SOML read operation can be applied to different 3D NAND SSDs, we conducted sensitivity tests by varying: (1) DRAM capacity and (2) partial-page granularities.

DRAM capacity sensitivity: Figure 14a shows the throughput comparison between the baseline and our SOML read across various DRAM sizes. As can be observed, making the DRAM buffer bigger (from 64MB to 256MB) provides


(a) Different DRAM Capacity.


(b) Different partial-page granularities.

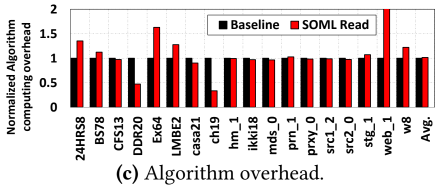

(c) Algorithm overhead.

Figure 14. Sensitivity tests and overhead comparisons between the baseline and the SOML read.

nearly no performance improvement. This is because a 64MB DRAM is very large; hence, the write latencies are successfully hidden, while the read performance cannot be further improved due to the non-repetitive access pattern of reads. In contrast, making the DRAM buffer smaller or not employing a DRAM buffer (0MB) degrade the overall performance. This is because the write latencies *cannot* be hidden by the DRAM buffer; as a result, the performance is dominated by the writes, instead of the reads. Hence, SOML read can only slightly improve the performance of the DRAMless SSDs.

Partial-read granularity sensitivity: Figure 14b shows the throughput comparison between the baseline and our SOML read across various partial-read granularities. As can be observed, the throughput of 4 partial-reads outperforms that of the other two granularities. This result stems from two reasons. First, more number of partial-read granularities demand the insertion of more number of additional SOML transistors, which in turn increases the read latency, ultimately degrading the overall performance. Second, fewer workloads are dominated by 1K or 2K read requests (shown in Figure 3b); as a result, 1/8 partial-page (2K) read cannot be easily utilized.

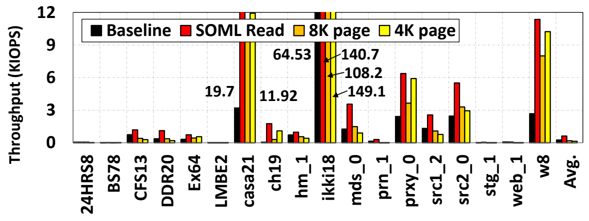


Figure 15. The comparison between SOML read and smaller pages.

6.2.4 Computation Overhead

To construct a SOML read operation, we propose Algorithm 1 to linearly search feasible reads in the request queue. However, the proposed algorithm has a higher time complexity compared to the baseline request selection algorithm, which always chooses the first read request. Figure 14c shows the normalized total computation times of the baseline and our algorithm. As can be observed, on average, our algorithm takes 1.13x longer time, which, in our opinion, is negligible. This is because the request computation time is much smaller than the latency of the NAND operation; hence, the computation time can be successfully hidden, while NAND chips being read/written.

6.2.5 Smaller page sizes

One may think that reducing the page sizes to half or quarter can solve the performance degradation caused by the larger page size (16KB). Figure 15 plots the throughput comparison between our SOML read and two smaller page sizes, 8K and 4K. Note that, to have a fair comparison, we make the same densities across all settings via increasing the number of pages per block 2x and 4x for 8K and 4K page sizes, respectively. As can be seen, 4k and 8K page sizes can outperform the baseline under some workloads (such as w8 and prxy_0) due to the reduced resource conflicts across chips. However, under some workloads (such as hm_1), the baseline outperforms the SSDs with reduced page sizes, since smaller page sizes reduce the overall throughput. In contrast, SOML read can improve the overall performance by increasing the intra-chip parallelism.

7 Related Work

7.1 Read Performance Enhancement Proposals:

We are not aware of any prior work that targets improving the intra-chip read parallelism in 3D NAND flash; so, we compare our proposed scheme against the existing 2D NAND flash read performance enhancing techniques.

Re-Read related proposals: The 2D NAND proposals use various techniques to minimize the number of re-reads required for servicing a read request. The studies in [6–9, 29] characterized the disturbances of the 2D MLC NAND flash in-detail. By using such characterization data, SSDs can correctly guess the NAND cell reliability status, so that a minimal number of re-reads is required for each read request. Zhao et al. [49] proposed a progressive voltage sensing strategy, which allows the number of re-reads to be varied based on the reliability of individual pages, instead of the worst page. As a result, the number of re-reads can be minimized.

Page disparity aware proposals: Liu et al. [31] proposed techniques to record the errors in pages, in an attempt to utilize such information to accelerate the speed of error correction, thereby improving the overall read performance. In comparison, Chang et al. [10] proposed utilizing the asymmetric read latency property of the MLC NAND cell, so that the frequently-read data can be placed into faster pages to improve the overall read performance.

These proposals are orthogonal to our SOML read operation; therefore, they can be combined, if desired, with our SOML read operation to further improve the read performance.

7.2 Request Scheduling Proposals:

We are not aware of any prior request scheduling algorithm designed to improve the read intra-chip parallelism. Consequently, we contrast our SOML read request scheduling algorithm with general SSD request scheduling algorithms.
Mitigating inter-chip workload imbalance proposals: In a multi-chip SSD architecture, different chips may experience imbalanced loads, which in turn reduces the overall performance. This is because the requests may wait to be serviced by heavily-loaded chips, while the other chips remain idle. Dynamic write request dispatch [12, 15, 41, 42] redistributes the write requests, which are queued in a heavily-loaded chip, to other chips, so that the loads across chips could be balanced.
Garbage collection (GC) related proposals: GC involves a very large number of read/write operations to migrate the valid data from the victim blocks to other blocks. Foreground GC, which stalls all queued requests, can incur severe performance penalties. Therefore, partial or background GCs [18, 46] are introduced to distribute or schedule those GC-related read/write operations to idle times; as a result, the requests are not stalled and can be serviced as usual.

8 Conclusion

Due to the high storage capacity demands from the storage market, 3D NAND density keeps increasing. Unfortunately, high-density SSDs end up achieving lower multi-chip parallelism than their low-density counterparts. From our extensive workload analysis with varying number of chips, we found that the read performance degrades much more than the write performance when employing fewer chips. Therefore, to mitigate such performance degradation, we proposed a novel SOML read operation for 3D NAND flash, which can perform multiple partial-reads to different pages simultaneously. A corresponding SOML read scheduling algorithm (for FTL) is also proposed to take full advantage of the SOML read. Our experiments with various workloads indicate that, on average, the overall performance of our SOML read-enabled system with 8 chips outperforms that of the baseline with 16 chips. Further, our experiments also indicate that the proposed approach outperforms three state-of-the-art optimization strategies.

Acknowledgments

This research is supported by NSF grants 1439021, 1439057, 1409095, 1626251, 1629915, 1629129 and 1526750, and a grant from Intel. Dr. Jung is supported in part by NRF 2016R1C1B2015312, DOE DE-AC02-05CH 11231, IITP-2017-2017-0-01015, NRF-2015M3C4A7065645, and MemRay grant (2015-11-1731).

References

[1] 2013. Fatcache: memcached on SSD. https://github.com/twitter/fatcache. (2013).
[2] 2014. ONFI 4.0 Specification. http://www.onfi.org/. (April 2014).
[3] 2018. Bitcoin. https://bitcoin.org/en/. (Aug 2018).
[4] 2018. Micron 3D NAND flyer. https://www.micron.com/~/media/documents/products/product-flyer/3d_nand_flyer.pdf. (Aug 2018).
[5] 2018. RAIN. https://www.micron.com/~/media/documents/products/technical-marketing-brief/brief_ssd_rain.pdf. (Aug 2018).
[6] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2012. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 521–526.
[7] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. [n. d.]. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 438–449.
[8] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu. 2015. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 551–563.
[9] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. [n. d.]. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 123–130.
[10] D. W. Chang, W. C. Lin, and H. H. Chen. 2016. FastRead: Improving Read Performance for Multilevel-Cell Flash Memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (Sept 2016), 2998–3002.
[11] Hyeokjun Choe, Seil Lee, Seongsik Park, Sei Joon Kim, Eui-Young Chung, and Sungroh Yoon. 2016. Near-Data Processing for Machine Learning. http://arxiv.org/abs/1610.02273. (2016).
[12] Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T. Kandemir, Chita R. Das, and Myoungsoo Jung. 2017. Exploiting Intra-Request Slack to Improve SSD Performance. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. 375–388.
[13] Sumitha George, Minli Liao, Huaipan Jiang, Jagadish B. Kotra, Mahmut Kandemir, Jack Sampson, and Vijaykrishnan Narayanan. 2018. MDACache:Caching for Multi-Dimensional-Access Memories. In *The 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*.
[14] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. 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)*.
[15] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the international conference on Supercomputing (SC)*.
[16] Jae-Woo Im, Woo-Pyo Jeong, Doo-Hyun Kim, Sang-Wan Nam, Dong-Kyo Shim, Myung-Hoon Choi, Hyun-Jun Yoon, Dae-Han Kim, You-Se Kim, Hyun-Wook Park, and others. 2015. 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.
[17] Dawoon Jung, Jeong-UK Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. 2010. Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme. *ACM Transactions on Embedded Computing Systems* (March 2010).

[18] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir. 2014. HIOS: A host interface I/O scheduler for Solid State Disks. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*.

[19] 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. 2016. 7.1 256Gb 3b/cell V-NAND flash memory with 48 stacked WL layers. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. https://doi.org/10.1109/ISSCC.2016.7417941

[20] 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. 2017. 11.4 A 512Gb 3b/cell 64-stacked WL 3D V-NAND flash memory. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. https://doi.org/10.1109/ISSCC.2017.7870331

[21] Wonjoo Kim, Sangmoo Choi, Junghun Sung, Taehee Lee, C. Park, Hyoungsoo Ko, Juhwan Jung, Inkyong Yoo, and Y. Park. 2009. Multi-layered Vertical Gate NAND Flash overcoming stacking limit for terabit density storage. In *2009 Symposium on VLSI Technology*.

[22] O. Kislal, M. T. Kandemir, and J. Kotra. 2016. Cache-Aware Approximate Computing for Decision Tree Learning. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[23] Jagadish Kotra, D. Guttman, Nachiappan. C. N., M. T. Kandemir, and C. R. Das. 2017. Quantifying the Potential Benefits of On-chip Near-Data Computing in Manycore Processors. In *Proceedings of 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[24] Jagadish Kotra, S. Kim, K. Madduri, and M. T. Kandemir. 2017. Congestion-aware memory management on NUMA platforms: A VMware ESXi case study. In *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*.

[25] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das. 2016. Re-NUCA: A Practical NUCA Architecture for ReRAM Based Last-Level Caches. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[26] Jagadish B. Kotra, Haibo Zhang, Alaa Alameldeen, Chris Wilkerson, and Mahmut T. Kandemir. 2018. CHAMELEON: A Dynamically Reconfigurable Heterogeneous Memory System. In *The 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*.

[27] Miryeong Kwon, Jie Zhang, Gyuyoung Park, Wonil Choi, David Donofrio, John Shalf, Mahmut Kandemir, and Myoungsoo Jung. 2017. TraceTracker: Hardware/Software Co-Evaluation for Large-Scale I/O Workload Reconstruction. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*.

[28] 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. 2018. 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)*.

[29] Chun-Yi Liu, Yu-Ming Chang, and Yuan-Hao Chang. 2015. Read Leveling for Flash Storage Systems. In *Proceedings of the 8th ACM International Systems and Storage Conference (SYSTOR '15)*. New York, NY, USA.

[30] Jun Liu, Jagadish Kotra, Wei Ding, and Mahmut Kandemir. 2015. Network Footprint Reduction Through Data Access and Computation Placement in NoC-based Manycores. In *Proceedings of the 52Nd Annual Design Automation Conference (DAC)*.

[31] R. S. Liu, M. Y. Chuang, C. L. Yang, C. H. Li, K. C. Ho, and H. P. Li. 2016. Improving Read Performance of NAND Flash SSDs by Exploiting Error Locality. *IEEE Trans. Comput.* (April 2016), 1090–1102.

[32] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu. 2018. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[33] 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. 2018. A 512Gb 3b/Cell 3D flash memory on a 96-word-line-layer technology. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*.

[34] Alessia Marelli Rino Micheloni, Luca Crippa. 2010. *Inside NAND Flash Memory.* Springer Netherlands.

[35] Samsung. 2018. Samsung Pro 950 SSD. https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-950-pro-nvme-512gb-mz-v5p512bw/. (Aug 2018).

[36] Samsung. 2018. Samsung Pro 960 SSD. http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/960pro/. (Aug 2018).

[37] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. [n. d.]. Micro-pages: Increasing DRAM Efficiency with Locality-aware Data Placement. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS XV)*. 219–230.

[38] X. Tang, M. Kandemir, P. Yedlapalli, and J. Kotra. 2016. Improving bank-level parallelism for irregular applications. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12.

[39] Xulong Tang, Orhan Kislal, Mahmut Kandemir, and Mustafa Karakoy. 2017. Data Movement Aware Computation Partitioning. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17)*. New York, NY, USA, 730–744.

[40] X. Tang, A. Pattnaik, H. Jiang, O. Kayiran, A. Jog, S. Pai, M. Ibrahim, M. T. Kandemir, and C. R. Das. 2017. Controlled Kernel Launch for Dynamic Parallelism in GPUs. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 649–660.

[41] Arash Tavakkol, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2014. Unleashing the Potentials of Dynamism for Page Allocation Strategies in SSDs. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '14)*. 551–552.

[42] Arash Tavakkol, Pooyan Mehrvarzy, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2016. Performance Evaluation of Dynamic Page Allocation Strategies in SSDs. *ACM Trans. Model. Perform. Eval. Comput. Syst.* (June 2016), 7:1–7:33.

[43] Guanying Wu and Xubin He. 2012. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*.

[44] Qin Xiong, Fei Wu, Zhonghai Lu, Yue Zhu, You Zhou, Yibing Chu, Changsheng Xie, and Ping Huang. 2017. Characterizing 3D Floating Gate NAND Flash. In *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '17 Abstracts)*. ACM, 31–32.

[45] 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. 2017. 11.1 A 512Gb 3b/cell flash memory on 64-word-line-layer BiCS technology. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 196–197.

[46] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. 2017. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*. 15–28.

[47] P. Yedlapalli, J. Kotra, E. Kultursay, M. Kandemir, C. R. Das, and A. Sivasubramaniam. 2013. Meeting midway: Improving CMP performance with memory-side prefetching. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[48] Chun yi Liu, Jagadish Kotra, Myoungsoo Jung, and Mahmut Kandemir. [n. d.]. 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, 67–82.

[49] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*. USENIX.

[50] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E. Priebe, and Alexander S. Szalay. 2015. FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, 45–58.