PEN: Design and Evaluation of Partial-Erase for 3D NAND-Based High Density SSDs

Chun-yi Liu and Jagadish Kotra, *The Pennsylvania State University*;
Myoungsoo Jung, *Yonsei University*; Mahmut Kandemir, *The Pennsylvania State University*

https://www.usenix.org/conference/fast18/presentation/liu

This paper is included in the Proceedings of the
16th USENIX Conference on File and Storage Technologies.
February 12–15, 2018 • Oakland, CA, USA

ISBN 978-1-931971-42-3

Open access to the Proceedings of
the 16th USENIX Conference on
File and Storage Technologies
is sponsored by USENIX.

PEN: Design and Evaluation of Partial-Erase for 3D NAND-Based High Density SSDs

Chun-Yi Liu, Jagadish B. Kotra, *Myoungsoo Jung, Mahmut T. Kandemir
{cql5513, jbk5155, kandemir}@cse.psu.edu, *mj@camelab.org
The Pennsylvania State University, *Yonsei University

Abstract

3D NAND flash memories promise unprecedented flash storage capacities, which can be extremely important in certain application domains where both storage capacity and performance are first-class target metrics. However a block of 3D NAND flash contains many more pages than its 2D counterpart. This increased number of pages-per-block has numerous ramifications such as the longer erase latency, higher garbage collection costs, and increased write amplification factors, which can collectively prevent the 3D NAND flash products from becoming the mainstream in high-performance storage domain. In this paper, we introduce PEN, an architecture-level mechanism that enables partial-erase of flash blocks. Using our proposed partial-erase support, we also discuss how one can build a custom garbage collector for two types of flash translation layers (FTLs), namely, block-level FTL and hybrid FTL. Our experimental evaluations of PEN with a set of diverse real storage workloads indicate that the proposed approach can shorten the write latency by 44.3% and 47.9% for block-level FTL and hybrid FTL, respectively.

1 Introduction

NAND flash based solid state disks (SSDs) have become one of the dominant storage components in different computing domains, ranging from embedded systems to general purpose workstations to high-performance datacenters [6, 13, 22, 57]. High-performance computing employs SSDs in various ways such as an SSD cache [41, 55] or a bursty buffer [39], to mitigate the performance bottlenecks imposed by the conventional hard disk drives (HDDs).

While SSDs can significantly improve the overall system performance, there is also an emerging trend that pushes SSDs toward an entirely different direction [20, 31, 51]. Specifically, major flash vendors amplify storage capacity by transitioning from 2D NAND flash to 3D NAND flash. The 3D NAND flash technology layers flash cells vertically, which can increase the size of the individual flash dies. For example, Samsung stacks 100 layers of *charge trap flash* (CTF) cells, and as a result can achieve 1 Terabit density flash dies without any modification to the existing flash interface [10].

Layering multiple CTF cells increases the number of pages in a physical block, rather than the number of blocks within a die, which makes the internal micro-architecture of 3D NAND different compared to the 2D planar flash. Consider the vertical architecture of a particular 3D NAND flash [20], *VNAND*. VNAND increases the die density by stacking more layers, but in this architecture, CTF cells across the different layers share a same set of pillars (channel), thereby increasing the number of pages per block. Furthermore, as all the block-related control circuits of VNAND reside on the block decoder of the top layer due to staircase-like control gate [20], this decoder area controls the signals to all CTF cells of the underlying layers. Consequently, as one stacks more layers, the amount of such block-related control circuits for each block increases. However, the area from where one can control all layers is limited, which in turn reduces the number of blocks but increases the number of pages per block. Owing to this, a block of VNAND contains at least 3 times more pages per block compared to the 2D flash.

Unfortunately, the new structure of VNAND can exacerbate the overheads incurred due to garbage collection (GC), which is one of the well-known performance bottlenecks in modern SSDs. Specifically, the peripheral circuits and the micro-architecture of VNAND's [19, 26] large-granularity erase makes the latency characteristics of 3D NAND worse compared to 2D flash. In addition, the large number of pages per block can potentially accommodate more valid data that a flash firmware needs to migrate for each GC. The longer erase time and relatively more valid pages to migrate (i.e., a series of reads and writes) can have a significant performance impact on GC operations and may in turn render 3D flash difficult to directly replace 2D flash in many designs of high-performance SSD.

In this work, we propose *PEN*, a novel strategy to enable **P**artial **E**rase for 3D **N**AND flash technology. PEN

alleviates the GC overheads by introducing a finer erase unit in 3D NAND, which can reduce number of valid pages copied during a GC, thereby reducing the GC latency. To the best of our knowledge, this is the first work that investigates *partial-erase* for 3D NAND, starting from the circuit level and evaluating its architectural ramifications from both the performance and reliability angles. Our contributions can be summarized as follows:

- We present a comprehensive architectural support, "partial-erase operation" that addresses the potential performance degradation imposed by 3D NAND flash. The proposed low-level operation can selectively reset multiple pages instead of erasing a bulk block by modifying the 3D NAND peripheral circuits, page decoders, and command logics, with minimum area overhead.
- While our partial-erase operation can be leveraged to alleviate performance degradation, the number of pages per reset should be determined at the design time. However, it is a non-trivial task to statically decide the operation granularity of such partial-erase operation. This inflexibility in turn can lead to a large mapping table size or introduce more valid page migrations. We propose a novel GC algorithm, called "M-Merge" that keeps the original mapping table size when the partial-erase operation is employed. In addition, M-Merge can adaptively decide the optimal partial-erase granularity by considering program-disturbance issues at a run-time.
- We demonstrate that the performance degradation in 3D NAND flash stems from the increased number of valid pages copied during a GC operation. The evaluation studies using a set of 12 real storage workloads reveal that our PEN mechanism (putting partial-erase operation and M-Merge together) can significantly reduce the valid page copies during a GC operation, thereby improving the overall system performance.

2 Background

2.1 NAND Flash Organization

NAND flash memory consists of several blocks constituting a plane, as shown in Figure 1. Each block is made up of a number of pages. Page is a unit of read and write, and its size varies from 2KB to 32KB [1]. A traditional 2D NAND flash typically consists of 128 to 192 such pages per block. The number of pages per block increases [10] in a 3D NAND flash, and it can be as high as 576 [29].

Figure 1 shows a vertical-channel 3D NAND flash implemented by Samsung [20]. NAND flash cells connected in series form a pillar (channel) with top and bottom select transistors represented by UpperST and LowerST, respectively. Cells in the same horizontal axis form a page, represented by P-0, P-1, etc. The upper select

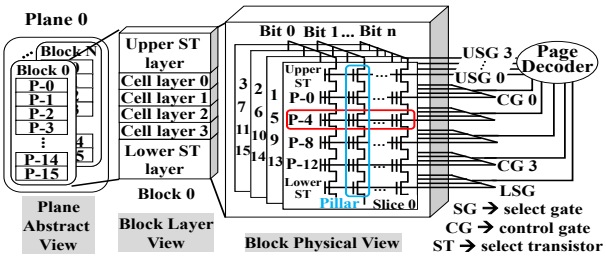


Figure 1: Vertical-channel NAND flash circuit.

gates (USG) and the control gates (CG) are used in accessing a page corresponding to an I/O request. USG is used to select the corresponding slice and the CG is used to select the corresponding page in a slice.

The basic operations in a NAND flash chip are read, write, and erase [46]. The erase operation resets the data to value "1" in a page. It is performed at a *block-granularity*; the data in all the pages in a block are reset per erase operation. In a NAND flash device, the number of erase cycles is limited. Typically, an erase operation is implemented in two phases: (1) data-erase and (2) erase-verify. During the data-erase phase, the data in all the pages in a victim block are reset, and the erase-verify phase checks if all the pages have been successfully reset or not. The entire erase operation is iteratively repeated until the data in all the pages are successfully reset.

The data-erase circuit implementation of an erase operation is vendor-specific. There are two popular implementations: (a) bulk data-erase (implemented in Samsung SSDs) [20] and (b) gate-induced drain leak (GIDL) (implemented in Toshiba and Macronix SSDs) [27, 31, 51] data-erase. Bulk data-erase operation imposes a high voltage (typically 20V) to the shared substrate (containing multiple blocks), while the CGs for the block being erased are set to 0V. Hence, all the pages in a block are erased per erase operation unlike the GIDL implementation. In GIDL, the data-erase operation is implemented at a pillar granularity, and all the pillars in the same block are erased simultaneously. More specifically, it is implemented by imposing high voltage to USGs, LSG and bitlines, while the voltage of CGs is set to different values based on the strength of GIDL. The erase-verify implementation is achieved by imposing CGs with 0V, while SGs are imposed with bypass voltage. An unsuccessfully erased page is identified by measuring the current passing through the channels when bitline voltage changes from 0V to floating.

2.2 Flash Translation Layer (FTL)

The NAND flash vendors implement a Flash Translation Layer (FTL) [3, 15, 18, 21, 38, 40] to keep track of the physical location of a page in flash chips. FTL imple-

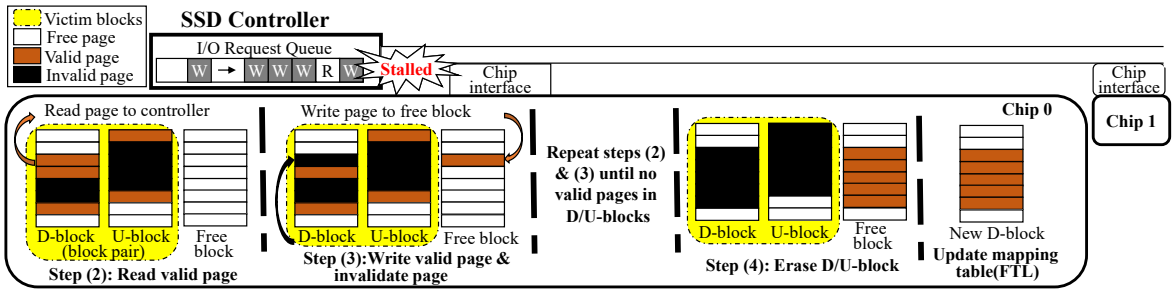


Figure 2: NFTL GC (Merge) overview.

ments two major functionalities, namely, address mapping and garbage collection.

2.2.1 Address Mapping

FTL maintains a data structure called mapping table. It maps a given logical page/block address to the physical location. Each read/write I/O request has to be translated by FTL to route the request to a corresponding physical page. When a logical page is updated, the old physical page is marked as "invalid," since SSDs do not support in-place update. The number of invalid pages in chip increases as the write requests are processed. Address mapping can be broadly classified into three categories based on the granularity at which the mappings are managed viz., (a) page-level, (b) block-level, and (c) hybrid mapping.

Page-level Mapping: This address mapping implementation needs a huge mapping table to manage translations at a page granularity. Note that a 1TB SSD requires at least 1GB mapping table. While SSD capacity doubles as the number of stacked NAND layers increases, such a huge mapping table becomes a major issue for SSD design (in terms of both price and power consumption).

Block-level Mapping: Block-level mappings only store the mapping information per block, and therefore, the size of mapping table is smaller than other mappings.

A well-known block-level implementation is NFTL [3]. In NFTL, the mapping information of a block consists of a Data block (D-block) and Update block (U-block). A D- and U-block together are referred to as a block-pair. D-block represents the actual data block where a page is originally mapped to, while U-block represents the block to which the updated pages from D-block are written to, leaving the corresponding paired D-block page invalid. Typically, the number of U-blocks is much smaller than the number of D-blocks; so, multiple D-blocks compete for a U-block.

A new write of NFTL is performed on the mapped page in the D-block, while an updated write to the same address is logged in the paired U-block. As a result, a read to an address may have to search (read) pages from the U-block sequentially to retrieve the latest copy. Hence, the read and write performance can be slower in NFTL compared to that of the page-level mapping.

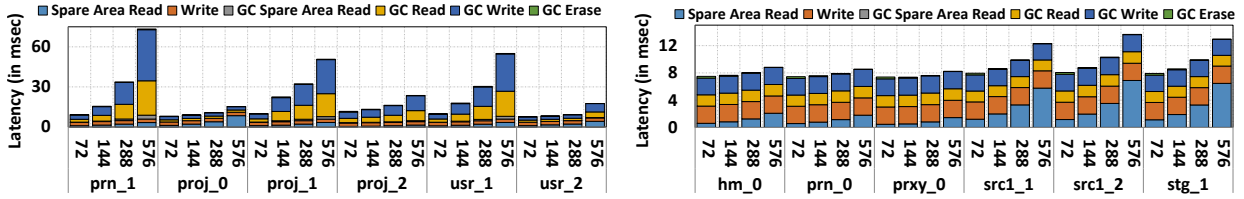
Hybrid Mapping: The hybrid mapping combines the best of the two previous mappings by (a) adopting the block-level mapping to reduce the mapping table size, and (b) utilizing partial (or small) amount of the page-level mapping table to accelerate the performance. Various such proposals include Superblock [21], FAST [40], DFTL [15], and LAST [38]. In this work, we only focus on the Superblock FTL implementation.[1]

2.2.2 Garbage Collection (GC)

GC is triggered by the write requests or the controller firmware to clear the invalid pages left in the flash chips, so that SSDs have enough free pages for the future writes. GC typically contains four steps: (1) selecting victim blocks, (2) reading valid pages from the victim blocks, (3) writing the valid pages into the reserved free blocks, and (4) erasing the victim blocks. Typically, steps (1) and (4) are executed only once, while steps (2) and (3) are executed repeatedly until no valid page is left in the victim blocks. Since GC changes the FTL address mapping, different FTLs implement their GC algorithms.

Figure 2 illustrates the GC in NFTL, which is mainly achieved by **Merge** operation. Merge *copies* all valid pages contained in victim block pair to a reserved free block, after which the victim block pair is erased. There are two scenarios when a GC will be triggered: (a) fully-utilized U-block and (b) unpaired D-block. In scenario (a), the paired U-block has no free page, and therefore, GC needs to be triggered to clear the invalid pages in this block pair. In scenario (b), the D-block corresponding to a write request does not have a U-block paired with it; as a result, GC is triggered on another block pair to reclaim a free U-block. We assume that the victim block pair for step (1) has already been selected. The second step involves reading the valid page from the victim block pair. In the third step, the page read in the second step is writ-

[1] Our proposal works equally well for other hybrid FTL implementations as well.



(a) High write-intensive workloads

(b) Low write-intensive workloads

Figure 3: Write latency breakdown for different block sizes in the case of NFTL.

ten into the free block and the read page is invalidated in the victim block. These steps, (2) and (3), are repeated for all the valid pages in the victim blocks, as depicted in the figure. Once all the valid pages are copied, in step (4), all the pages in the block are erased using an erase operation, also shown in the figure. Finally, the FTL address mapping is updated to reflect the new D-block.

As depicted in Figure 2, the on-demand read/write I/O requests *cannot* be served by the SSD chips during the GC and are stalled in the per-channel DRAM queue [4, 32–36, 43, 48, 52, 56] in the SSD controller. As a result, the GC can negatively affect the application performance [23–25]

2.3 Effect of Block Size

2.3.1 Effect of Block Size on Performance

With the increase in density for the 3D NAND flash, the number of valid pages per block increases. As a result, the number of pages to be copied from the victim block to the free block during a GC also increases. Consequently, the GC duration increases, in turn increasing the access latency for the read and write I/O requests, thereby degrading the overall performance significantly. This issue is widely referred to as the "Big Block" problem [54].

We performed experiments to quantitatively demonstrate the relationship between GC and the number of pages per block in the case of NFTL (block-level FTL)[2]. All the configurations tested have the same SSD capacity to prevent the capacity from affecting the GC triggered frequency. To keep the same capacity, we ensure that a plane has the same number of pages, but the number of pages per block are varied across different configurations. Here are the four evaluated configurations (blocks per plane, **pages per block**): (a) (15104, **72**), (b) (7552, **144**), (c) (3776, **288**), and (d) (1888, **576**). The rest of parameters can be found in Table 1. All four configurations are evaluated on the SSDSim [17] simulator using 12 write-dominant workloads shown in Table 2.

[2]We observed that Superblock FTL has a similar trend.

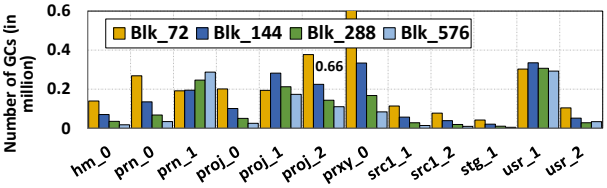


Figure 4: Effect of block sizes on the number of GC invocations in the case of NFTL.

Figures 3a and 3b show the breakdown of average write access latencies (in milliseconds) for the different block sizes. As can be observed, as the number of pages per block increases, the latency increase and becomes maximum for a block with 576 pages. The access latency incurred by a write request includes (1) the time spent in performing the actual write to the pages, and (2) the time spent in waiting in the I/O request queue if this write triggers a GC. The GC time which effects the wait time of a write I/O request can further be broken down into (a) GC spare-area read, (b) GC Read, (c) GC Write, and (d) GC Erase, as shown in Figures 3a and 3b. The GC spare-area read time accounts for the time spent in reading the page status to identify the valid pages. The GC read/write time accounts for copying the identified valid pages from the victim blocks to a free block, while the GC erase time accounts for the time spent in performing the erase of the victim blocks.

Figures 3a and 3b plot the write access latency breakdown for the high and low write-intensive workloads, respectively. In Figure 3a, as the number of pages per block increases, the time spent in copying the valid pages (which includes GC read/write) increases from 58% for a block with 72 pages to 79% for a block with 576 pages. This result indicates that reducing the number of valid pages to be copied is crucial under the high-intensive workloads. In figure 3b, the time spent in copying the valid pages remains the same, but the time spent in reading the spare-area increases due to the inherent design of NFTL.

Figure 4 shows the number of GC invocations for different block sizes for all workloads. In general, the number of GC invocations are halved as the number of pages

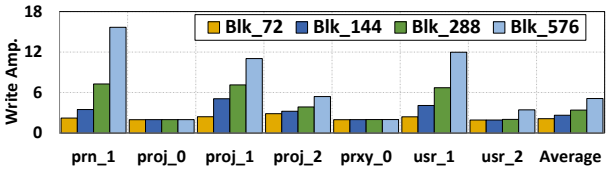


Figure 5: Effect of block sizes on writes amplification in the case of NFTL.

per block doubles; so, the time spent in erasing the blocks decreases. However, in the high write-intensive workloads, such as prn_1 and proj_1, the number of GCs increases as the number of pages per block doubles. This is because the configuration with a fewer number of blocks (larger block) has fewer competing blocks, thereby resulting in higher number of GCs.

2.3.2 Impact of Block Size on Lifetime

Write amplification factor is the ratio of the amount of data which the host writes and the amount of writes that actually occurs on the flash media (including the GC writes) [16].

Figure 5 shows the write amplification varying block sizes for high write-intensive workloads. Write amplification for the high write-intensive workloads increases, on average, from 2.1 for a block with 72 pages to 5.1 for a block with 576 pages. This is because, as the number of pages per block increases, the corresponding number of valid pages per block also increases. Since the erase operation during a GC is at a block-level, all the valid pages from the victim block need to be copied to a free block, thereby increasing the write-amplification. Such increased write-amplification may shorten the lifetime of SSDs. We also observed that Superblock FTL [21] has a similar trend. However, page-level FTLs do not suffer from the increased write-amplification because of smart page allocation and victim block selection algorithms [2, 12, 16, 42, 44, 45].

3 Overview of Partial-Erase Operation

Due to the large number of pages in a block, 3D NAND-based flash storage can be subjected to excessive valid page copy overhead. To reduce the number of valid page copies in 3D NAND, we propose a *partial-erase operation for 3D NAND flash* (PEN). Unlike the block-level erase operation, our proposed partial-erase operation performs erase at a "partial block" (PB) granularity. Since our proposal enables partial-erase, the amount of valid pages that need to be copied during a GC is reduced significantly, eventually improving the write latency and the overall I/O throughput. Also, since the number of pages copied during a GC reduces, the overall number of writes induced by GCs is also reduced, and this in turn results in lower write-amplification and increased lifetime.

Our proposal consists of both hardware and software changes. On the hardware side, our hardware modifications for the partial-erase operation support both the partial data-erase phase and the partial erase-verify phase. The implementation of the partial data-erase phase includes inhibiting the erase of pages other than the pages in a PB of a block. Similarly, the implementation of partial erase-verify phase includes only verifying the pages in PB to decide if the partial-erase operation needs to continue erasing the non-erased pages in PB or not.

The partial-erase operation may incur additional program disturbances to the neighboring pages, causing the neighboring cells' data to be modified. To solve the disturbance, we provide a software-based modification, since the hardware-based solutions [11, 14] typically reduce the 3D NAND array density, which may trigger further GCs. Another reason is that the hardware-based solutions can only mitigate the disturbance, so the data in the boundary pages may still be corrupted with more partial-erase operations. Therefore, a software solution is more preferable.

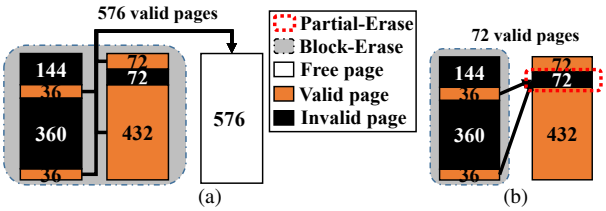
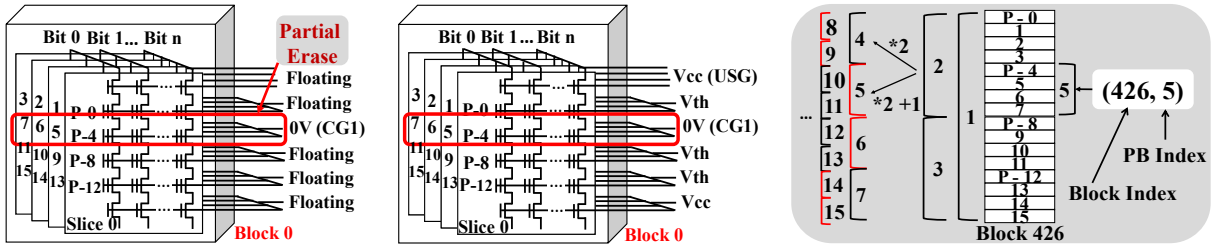


Figure 6: (a) depicts the baseline without partial-erase support necessitating 576 valid pages to be copied from the victim blocks to the new block. (b) depicts our partial-erase support which only necessitates 72 valid pages to be copied.

On the software side, we propose a modified merge (M-Merge) algorithm in GC that utilizes our partial-erase operation to reduce the number of valid pages copied from the victim blocks. One major contribution of our M-Merge algorithm is that, the valid pages in victim blocks are "consolidated" into fewer blocks, unlike in the baseline GC algorithm where all valid pages are copied to free blocks. Therefore, in our proposed algorithm, we copy very few pages during GC. Another contribution is that the proposed M-merge algorithm is aware of the possible disturbance by the partial-erase, so the data in non-erased pages will not be corrupted by disturbances. Figure 6 shows the difference between the baseline GC and M-Merge based GC. In this example, our M-Merge only copies 72 valid pages during a GC; in comparison, the baseline GC copies a total of 576 valid pages.



(a) Partial data-erase phase. (b) Partial erase-verify phase. (c) Partial block index.

Figure 7: Partial-Erase operation design.

4 Controller Hardware for Partial-Erase

4.1 Peripheral Circuit Modifications

The peripheral circuit modifications for our proposed partial data-erase and partial erase-verify phases are covered in this subsection. In the baseline erase implementations of VNAND (bulk-erase) circuits, the erase-inhibition technique is used to restrict the erase operation to the victim block. That is, only the CGs of the victim block are set to 0V, and the rest of the blocks are floating.[3] Therefore, only the cells in the victim block are under high negative voltage difference, which resets all the cells in the victim block. In contrast, in the data-erase phase of the partial-erase operation, the erase-inhibition is achieved at a partial-block (PB) granularity, as shown in Figure 7a. In other words, only CG1 of PBs are set to 0V, instead of the entire block as in the baseline bulk-erase implementation. Figure 7a shows the implementation of the partial data-erase for a PB with 4 pages in the 3D NAND flash. In this example, only pages P4-P7 are erased, while the other pages retain their earlier contents as their corresponding CGs are set to floating.

The partial erase-verify phase verifies if all the pages in the PB are erased successfully during the data-erase phase or not. A modified read operation can be used as partial erase-verify. Instead of selecting the page by only one USG and one CG for read operation, multiple USGs and CGs are used to select all pages in a PB to realize the partial erase-verify phase. Then, the current on the bitlines can be measured to figure out the PB cells' erasing status. Figure 7b shows the implementation details of proposed partial erase-verify phase. The partial erase-verify phase is performed on the second 4 pages (P4-P7). The CG1 is set to 0V, so that the content in cells of second 4 pages will reflect to the current on bitlines. The GIDL-based erase can be modified similarly.

The hardware modification for our partial-erase operation mainly adds an extra circuit for the control logic in the page decoder (PD), as illustrated in Figure 8.

[3]The CG is disconnected; so, the voltage of the cells becomes floating.

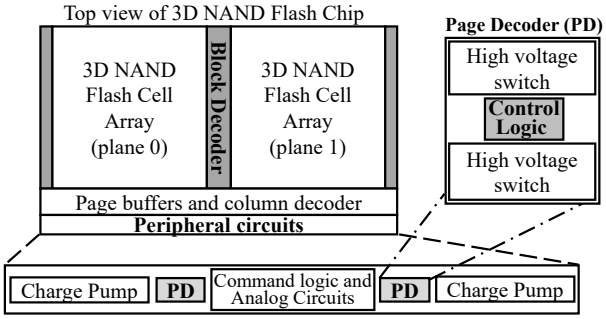


Figure 8: Partial-Erase circuit overview.

The required area for the control logic is at most the same as the original control logic area. The reason is that the proposed circuit has fewer cases to handle than the original one. The area overhead of the proposed enhancements can be calculated as follows: The peripheral circuits in 3D NAND flash chip are around $6 \sim 9\%$[19, 26, 29, 53], and the PD occupies about 4% of the peripheral circuits[47]. In the PD, only 17% of the area is devoted to the control logic[47]. Therefore, our hardware overhead is at most 0.07% of the whole area, which is negligible in a 3D NAND flash chip.

4.2 Boundary Program Disturbance

The additional program disturbance to the neighboring pages [5, 58] comes from later write (program) operation after the partial-erase. In Figure 7a, pages P0-P3 and P8-P11 represent the neighboring pages that are disturbed by the program operations after the partial-erase operation. Such 3D NAND program disturbance is known to be small compared to the 2D counterpart, owing to the 3D CTF cell itself [19]; so, the boundary pages can tolerate more extra program disturbance in 3D NAND compared to 2D. However, the data in those pages can still be corrupted if the same partial-block is repeatedly erased.

4.3 Indexing the Partial Blocks

We augment the original block-level erase operation command format shown in Figure 9 with partial-block

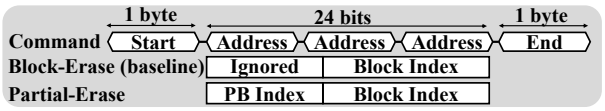


Figure 9: Partial-Erase command format.

index bits. These additional bits, which represent the PB index along with the already-existing block index bits, enable our proposed partial-erase operation.

To minimize the control circuitry and utilize the existing command convention and format used by the NAND flash, we restrict each PB to contain ρ number of pages, which can be expressed as:

$$\rho = \frac{\text{number of pages per block}}{2^l} \qquad \text{(PB size)}$$

,where l represents the number of times that a block is split into two smaller equal-sized partial-blocks. l can take values ranging from 0 to L, where L is the maximum number of times a block can be split. If l is 0, a block is *not* split causing the entire-block to be erased. If l equals to 1, an entire block is split into two partial-blocks so that a partial-erase can be performed on either partial-blocks (PBs). With restricted PB sizes and locations, the number of supported PBs for the chip can only be $2^{L+1}-1$. For example, if L is 6 in a chip containing 576 blocks, the number of pages in a PB can take the following values: 576, 288, 144, 72, 36, 18 and 9 pages. Hence, the possible PBs per block can be 127 $(1+2+4+8+16+32+64 = 2^{6+1}-1 = 127)$.

Figure 7c shows our PB indexing scheme. Instead of indexing a PB in a block with both PB size and location, we use a single PB-index to identify both the PB size and location. Thus, a tuple (block-index, PB-index) is used to identify the unique PB amongst multiple blocks in the 3D NAND flash. For example, PB $(426,2)$ contains two smaller PBs, PB $(426,4)$ and PB $(426,5)$. In addition, erasing one bigger PB is less expensive than erasing two corresponding smaller PBs, since the NAND chip can only execute one command at a time due to its internal control circuitry. As a result, the PB size used during GC highly affects the GC latency.

5 FTL for Partial-Erase

5.1 Partial Block vs. Smaller Block

The simplest approach to utilize partial-erase operation in current FTLs is to replace the block-erase directly with the partial-erase; so, the block size in new system shrinks to one of the possible PB size, which is fixed and cannot be dynamically changed. Although this option is feasible, it has two main drawbacks: (1) larger mapping table size and (2) fixed partial-erase granularity. Enlarging the mapping table size in modern block-level and hybrid FTLs are costly. For example, in the extreme case, where L is 6, 2^6 times of baseline mapping table size is required. On the other hand, employing a fixed partial-erase granularity can result in sub-optimal performance as:

• A finer fixed partial-erase granularity might necessitate more number of partial-erase operations to reclaim a large number of invalid pages in the victim block. These partial-erase overheads can be reduced by fewer coarser granularity partial-erase operations.

• A coarser fixed partial-erase granularity might result in more number of valid page copies.

Motivated by this, we introduce our M-merge algorithm which can keep original mapping table size and dynamically choose the optimal partial-erase granularity.

5.2 M-Merge for Block-level Mapping

We propose a new GC algorithm, **M-Merge**, for NFTL. Figure 10 shows the difference between the baseline Merge algorithm and our proposed M-Merge algorithm. Our M-Merge algorithm is based on a sub-operation called restore. **Restore** is an operation which is performed on one of the PBs in D-block and its corresponding valid pages in U-block. It is composed of three stages: (1) valid-page copy from D-block to U-block , (2) partial-erase of PB in D-block, and (3) valid-page copy from U-block to D-block. In the first stage, all the valid pages in the PB must be copied to the U-block or other blocks, so that a partial-erase can be performed for the PB which has only invalid and free pages in the second stage. After the PB is partial-erased, the third stage copies back the valid pages that belong to this PB from U-block. The cost of the restore operation is one partial-erase and several valid page copies corresponding to this PB. The cost of the restore operations in Figure 10 can be calculated as follows:

$$\begin{aligned} restore(PB\ 5) &= None(skipped) \\ restore(PB\ 9) &= 1\text{ PE} + 72\text{ page copies} \\ restore(PB\ 14) &= 2\text{ page copies} + 1\text{ PE} + 72\text{ page copies} \end{aligned}$$

5.2.1 M-Merge Examples

In the example shown in Figure 10, assuming that there are a total of 576 (434 in D-block + 142 in U-block) valid pages, when a GC is triggered, all these valid pages from D-block and U-block need to be copied to a free block. Hence, the cost of a GC merge operation in the baseline is the cost of copying these 576 pages to a free block along with the cost of erasing both the D- and U-blocks. However, our M-Merge algorithm uses two restore sub-operations to achieve the same goal.

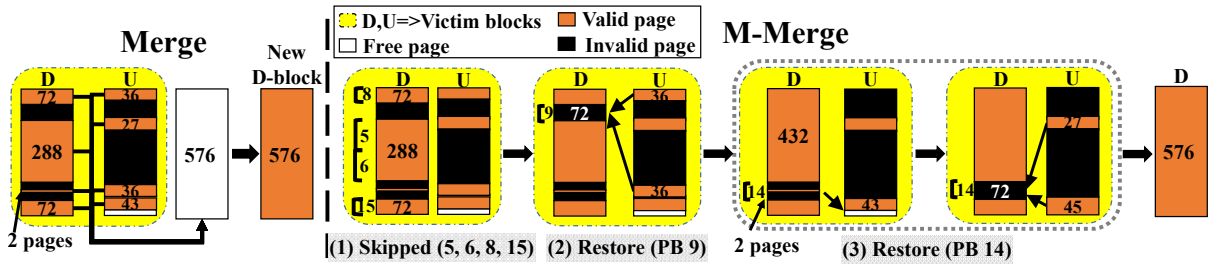


Figure 10: Merge and M-Merge operations.

M-Merge only executes the restore operations on D-blocks, while U-blocks are still block-erased in the end. We use D-block PB indices (Figure 7c) of 8, 9, 5, 6, 14, and 15 to explain M-merge. Note that the non-overlapped PBs form a complete block; hence, restoring all the non-overlapped PBs can guarantee that the U-block is erasable. We arrive at these PB indices by applying our Algorithm 1, which will be discussed shortly. As can be observed from Figure 10, PBs 8, 5, 6 and 15 have no invalid pages; so, M-Merge skips those PBs. PBs 9 and 14 contain invalid pages, so two restore operations need to be performed. PB 9 has only invalid pages, and hence, the first stage of the restore operation is skipped, and only the following two stages are executed. Thus, the corresponding 72 (36 + 36) valid pages in the U-block are copied back to PB 9. On the other hand, PB 14 has two valid pages, and as a result, those two pages have to be copied to the U-block in the first stage, and then the last two stages can be performed aiming PB 14. Overall, the total cost of M-Merge is the time taken for copying 72 + 2 + 72 = 146 valid pages, partial-erasing 2 PBs, and erasing 1 U-block.

Typically, a NAND flash read/write is 10 times faster than an erase operation [1]. The speedup brought by our proposed M-Merge operation over the conventional Merge operation is:[4]

$$speedup = \frac{(Merge\ time)}{(M\text{-}Merge\ time)} = \frac{(576+20)}{(146+20+10)} = 3.38\times.$$

M-Merge with Program Disturbance: We assume that the previous example executes 4 times; as a result, PB 9 is restored 4 times by M-merge, which is shown in Figure 11. As discussed in Section 4.2, the pages adjacent to PB 9 are disturbed when PB 9 is restored. To prevent the data corruptions caused by the disturbances, those disturbed pages will be restored by the next subsequent M-Merge operation that disturbs those pages. Note that the smallest PB has only 9 pages. At time $t1$, the PB 9 is restored. Thus, the upper PB (9 pages) and the lower PB (9 pages) are disturbed, but the data in those two layers are still consistent (not corrupted). At time $t2$, although only PB 9 is required to be restored, the disturbed pages (upper and lower PBs) are restored as well, since the data in these PBs may be corrupted due to this partial-erase operation. Hence, $9+72+9$ pages are restored. At time $t3$, only PB 9 is restored, since both upper and lower PBs are corrected by the previous restore operation at time $t2$. At time $t4$, two upper PBs, two lower PBs and PB 9 are restored to prevent the data corruption.

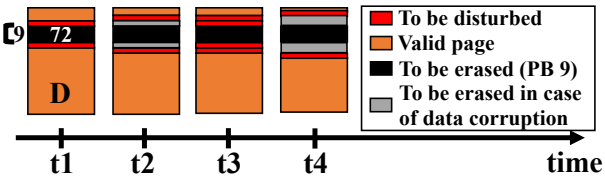


Figure 11: Data corruption prevention.

In summary, M-merge restores the possibly corrupted boundary pages to prevent the data corruption. Although this proposal increases the number of copied pages during M-merge, it still outperforms the baseline merge algorithm, which can be observed in Figure 15a.

M-merge wear-leveling: In the example shown in both Figure 10 and Figure 11, the PBs in the same block may be under different number of erase operations due to partial-erase operation. To mitigate wear-unleveling, the approach in [30] is adopted. To be more specific, a block can only be M-merged W times before a baseline Merge, where W is a (preset) wear-leveling parameter.

5.2.2 M-Merge Algorithm

$$cost[pb] = Min(restore(pb, disturb), \\ cost[pb*2] + cost[pb*2+1]) \quad (1)$$

Our M-Merge algorithm can be accomplished through various sequences of restores. However, the sequence that yields the "minimum" cost and preventing data corruption is preferred. To find such sequence, the recursive relationship in Equation (1) is introduced, which estimates the cost, represented by $cost(pb)$, of using restore for PB index (pb), comparing it with the total cost incurred for the PB indices ($pb*2$) and ($pb*2+1$). Then,

[4]The computation time can be omitted, since the execution time of NAND flash erase operation is on a millisecond scale.

the one which offers the minimum cost is chosen. Note that the restore operation is aware of disturbance, so all of the valid pages in the PB may be copied out and back in case of a data corruption, even though the PB has no invalid pages.

When a GC is triggered, the total cost for M-Merge operation is recursively estimated using Equation (1) starting with the whole block, which is PB 1. The estimated total cost for M-Merge operation is compared with that of baseline Merge operation. Based on the relative costs, the less expensive option is chosen. Note that, *our GC algorithm can switch between Merge and M-Merge operations dynamically based on the relative costs*, thus it is highly adaptive.

Algorithm 1: M-MERGE PLAN ALGORITHM

```
   Input: Dblk: D-block, dis: disturbed pbs in D-block
1  for pb ← max_pb to 1 do                                   // All PBs
2      cost[pb] ←RESTORE(Dblk, pb, dis);
3      trav[pb] ← leaf_PB;
4  for pb ← (max_pb/2) to 1 do              // Except the smallest PBs
5      if cost[pb∗2]+cost[pb∗2+1] < cost[pb] then // Equation (1)
6          cost[pb] ← cost[pb∗2]+cost[pb∗2+1];
7          trav[pb] ← internal_PB;
8  RstrSeq ← DFS-TRAVERSAL-LEAF-PB(cost, trav, 1);
9  toCopy ← SUM-OF-COPY(RstrSeq);
10 return (cost[1], RstrSeq, toCopy)
```

Algorithm 1 gives the pseudo-code that determines the minimum-cost restore sequence to perform M-Merge. After estimating the cost for M-Merge, the number of copied pages, which represents the number of pages to be copied in the first stage of all the restore operations, is calculated.

Algorithm 2: NFTL MERGE MODIFICATION

```
   Input: Dblk, Ublk
1  dis ← φ; corrupt ← True;
2  cost ← MERGE-COST(Dblk);
3  do                                                    // M-Merge cost
4      (mcost, RstrSeq, toCopy) ← M-MERGE-PLAN(Dblk, dis);
5      corrupt ← DISTURB-UPDATE(Dblk, RstrSeq, dis);
6  while corrupt ≠ True;
7  free_u ← FREE-PAGE(Ublk);
8  if free_u < toCopy then
9      (space, pb_u) ← LARGEST-INVALID-PB(Ublk);
10     mcost += PARTIAL-ERASE-TIME(pb_u);
11 if mcost < cost && toCopy < free_u + space &&
     Dblk.mmerge_count < W then                          // M-Merge
12     if free_u < toCopy then
13         PARTIAL-ERASE(pb_u);
14     for pb in RstrSeq do
15         DO-RESTORE(pb);
16 else                                                 // Baseline Merge
17     MERGE(Dblk);
```

Algorithm 2 presents the modifications proposed for the NFTL Merge operation. It initially calculates the costs for both the Merge and M-Merge operations, and then chooses the one with the lower cost. To prevent potential data corruption due to M-merge, Algorithm 2 is repeatedly called with the updated *PB*s' disturbance information. If NFTL decides to execute M-Merge based on the *toCopy* pages returned from Algorithm 1, a partial block in U-block may be erased to ensure that the restore sequence can be executed successfully.

M-Merge can generate the optimal restore sequence with minimum cost using Algorithm 1; however, executing the restore sequence is not guaranteed to be optimal due to insufficient free pages in U-block. In Algorithm 2, to provide more free pages for restore operations, we partial-erase the largest PB with all invalid pages before any restore operations. However, this PB in U-block may not be the optimal one, since a bigger PB may be generated during the execution of the restore sequence, not before it. In addition, the free pages in D-block and the newly-allocated block can also be used as temporary free pages. However, looking for these possibilities would take too much compute time. Hence, we choose the method in Algorithm 2.

5.3 M-Merge for Hybrid Mapping

Before describing our modifications to Superblock FTL, we briefly go over Superblock FTL [21]. In a superblock implementation, several adjacent logical blocks (say M), which is the basic unit of address mapping, are grouped to form a *superblock*, and each superblock contains several (physical) blocks (say N). The GC of Superblock FTL also employs the Merge operation at a block granularity. The GC for a superblock can be divided into intra- and inter-superblock GC.

Intra-superblock GC is triggered when a superblock has no free pages. The goal of the intra-superblock GC is to clear some free blocks for the subsequent write requests. Therefore, only the blocks with the minimum valid pages are merged by GC, and consequently, the read/write requests will not be stalled for too long.

Inter-superblock GC is triggered when there is no available free block in the NAND chip. The goal of the inter-superblock GC is to compact the victim superblock to the fewest number of physical blocks, which has only M physical blocks, so that the other superblocks can allocate available free blocks.

Superblock FTL can apply modifications similar to NFTL to reduce valid page copies by applying our M-Merge algorithm multiple times to physical blocks in a superblock. However, since the concept of D-block in the Superblock FTL is the cold data block (not the blocks to be restored), our modification must choose the D- and U-blocks using by M-Merge amongst the physical blocks in a superblock. Another important difference between NFTL and Superblock FTL M-Merge implementations

Algorithm 3: SUPERBLOCK MERGE MODIFICATION

```
Input: blkset: superblock physical block index set
cost ← SUPERBLOCK-MERGE-COST(blkset);
U-blkset ← BLK-SET-WITH-MIN-VALID-PAGES(blkset, M−N);
D-blkset ← blkset − U-blkset;
mcost ← 0;
for b in D-blkset do                                   // M-Merge cost
    dis ← φ; corrupt ← True;
    do
        (mcost_b, RstrSeq_b, toCopy_b) ← M-MERGE-PLAN(b, dis);
        corrupt ← DISTURB-UPDATE(b, RstrSeq, dis);
    while corrupt ≠ True;
    mcost += mcost_b;
if cost < mcost && blkset.mmerge_count ≤ W then // Baseline Merge
    SUPERBLOCK-MERGE(blkset);
else                                                   // M-Merge
    for b in D-blkset do
        if FREE-PAGE(U-blkset) < toCopy_b then
            ALLOC-FREE-BLOCK(U-blkset);
        for pb in RstrSeq_b do
            DO-RESTORE(pb);
```

is that the restore operation in Section 5.2 assumes the pages in D-block are in address order. However, they are out-of-order across the physical blocks in a superblock; as a result, the cost to perform restores needs to be adjusted accordingly.

Algorithm 3 gives the pseudo-code to determine whether it is beneficial to execute M-Merge and, if it is, how to perform M-Merge. The first step of Algorithm 3 is to decide *U-blkset* – the blocks to be erased. We pick $M-N$ blocks in the victim superblock with the minimum number of valid pages as *U-blkset*, so that the number of valid pages to be copied is minimal. The second step involves applying Algorithm 1 to all the blocks in *D-blkset* and estimating the total cost of M-Merge. The final step involves deciding whether it is beneficial to perform M-Merge or conventional Merge.

6 Evaluation

6.1 Experimental Setup

We used the "Flash core cell" model from HSIM [49] package in Synopsys HSPICE [50] to measure the partial-erase latency (in milliseconds) for a 3D NAND flash. Since the partial-erase latencies are governed by the cell with the slowest erase-rate and not by the number of cells per partial-block, the partial-erase latencies are only slightly better compared to the block-erase latency. Please refer to Table 1 for the details on the latencies. We used SSDSim [17] to evaluate our proposed M-merge algorithm for NFTL and Superblock FTL. Various parameters used in our SSDSim experiments are listed in Table 1. The read/write and block-level erase access latencies used in our SSDSim simulations are based on our empirical evaluations of the *real* 3D NAND chips. The 12 evaluated I/O workloads, whose characteristics are shown in Table 2, are from the revised SNIA traces [37].

SSD parameters	
(Page-read, Page-program, Block-erase)	(70μs, 900μs, 10ms)
(PB size (pages), partial-erase time (ms)) (L=6)	(288, 9.95), (144, 9.79), (72, 9.62), (36, 9.48), (18, 9.37), (9, 9.27)
(Channels, Chips, Dies, Planes, Blocks, Pages)	(8, 2, 2, 2, 1888, 576)
(Page Size, Spare area size)	(16KB, 1280B)
Total SSD capacity	1TB
(Over provision, Initial data)	(10%, 95%)
Number of tolerance disturbance	1
Wear leveling parameters (W)	16
NFTL parameters	
(Victim selection, GC free block threshold)	(Max invalid, 8%)
Superblock FTL parameters	
Logical:physical block ratio (M:N)	(4:5), (4:8)
Intra-/Inter-superblock GC threshold	(1 PBMT, 8%)

Table 1: Characteristics of the evaluated SSD.

trace	read reqs (in millions)	write reqs (in millions)	read data (in GBs)	write data (in GBs)	read coverage (in GBs)	write coverage (in GBs)
hm_0	1.417	2.576	9.96	20.47	1.84	1.63
prn_0	0.602	4.983	13.12	45.96	3.72	12.38
prn_1	8.464	2.77	181.35	30.78	73.78	11.52
proj_0	0.527	3.697	8.97	144.26	1.74	1.65
proj_1	21.143	2.497	750.36	25.57	693.5	9.03
proj_2	25.642	3.625	1015.9	168.68	409.37	115.13
prxy_0	0.384	12.135	3.04	53.8	0.29	0.7
src1_1	43.576	2.17	1485.6	30.34	116.69	4.16
src1_2	0.484	1.424	8.82	44.14	1.55	0.65
stg_1	1.4	0.796	79.52	5.98	79.42	0.39
usr_1	41.426	3.858	2079.2	56.12	651.16	24.56
usr_2	8.575	1.995	415.28	26.46	377.8	10.02

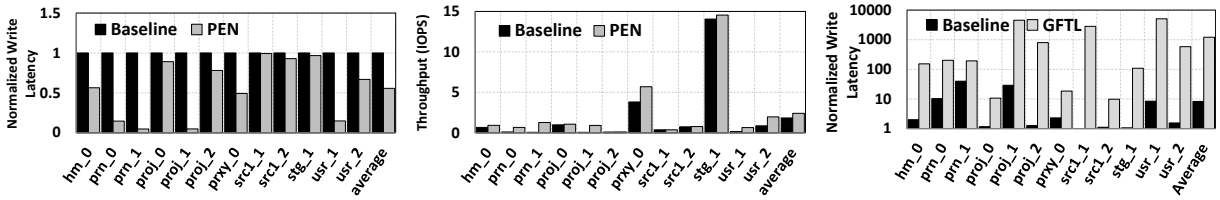
Table 2: Important characteristics of our workloads.

We use the following five metrics for our evaluations: (a) **average write latency**, (b) **throughput**, (c) **write amplification**, (d) **AEP**, and (e) **VEP**. AEP and VEP are the average and variance number of erase operations per page, respectively. Those two metrics track the number of erase operations at a finer granularity, page, unlike the coarse block granularity in the baseline. This is because, due to the partial-erase, the different pages in the same block can experience a different number of erases.

6.2 Experimental Results

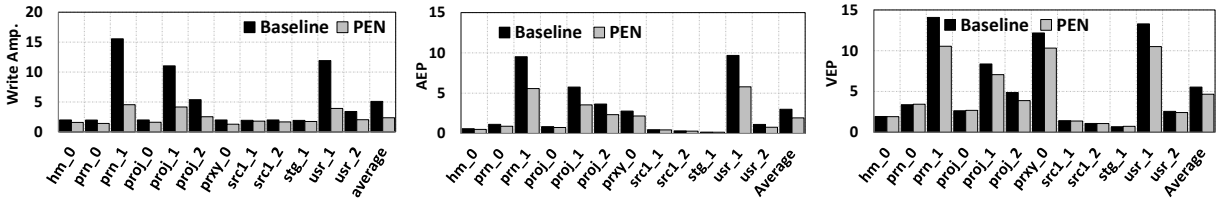
6.2.1 Block-Level FTL

Performance: Figure 12b shows the improvement in read/write throughput (IOPS) for our proposed PEN system (using partial-erase) over the baseline system (using block-erase only) for NFTL. Note that NFTL maintains the mapping at a unit of a block instead of a page; so, the GC trigger frequency is relevant not only to the ratio of the written data and the SSD capacity, but also to the page utilization of blocks; hence, a small amount of written data may trigger a large number of block merges.



(a) Normalized write latency for NFTL. (b) Throughput for NFTL. (c) Comparison with GFTL.

Figure 12: Performance improvements in the case of NFTL.



(a) Write amplification for NFTL. (b) AEP for NFTL. (c) VEP for NFTL.

Figure 13: Write amplification, AEP, and VEP improvements in the case of NFTL.

For example, although workload prn_1 only comprises of 30.78GB of written data, over 140,000 block merges are performed. Since M-merge can significantly reduce block merge overhead, the performance of the workloads with frequent block merge operations can be highly improved. On average, IOPS is improved by 1.43x over the baseline system in NFTL.

Figure 12a plots the write access latencies for our PEN system, *normalized* to the baseline system. The magnitude of improvement in the IOPS and write latencies is a function of the workload characteristics. For example, workloads like proj_1, usr_1, and prn_1, which have relatively high amounts of coverage (unique data), experience very high improvements. This is because, as the coverage in these workloads is very high, a majority of the U-blocks are already paired with a D-block resulting in an outage of free U-blocks. When a write is incurred to a page in an unpaired D-block, a block merge is triggered to reclaim a free U-block that can be paired with this D-block. Hence, for these workloads, the number of triggered block merges is very high, with each merge operation lasting several hundreds of milliseconds, owing to the increased number of valid page copies in the baseline. Since our M-Merge algorithm reduces the number of valid pages to be copied, our PEN system yields significant improvements in I/O throughput.

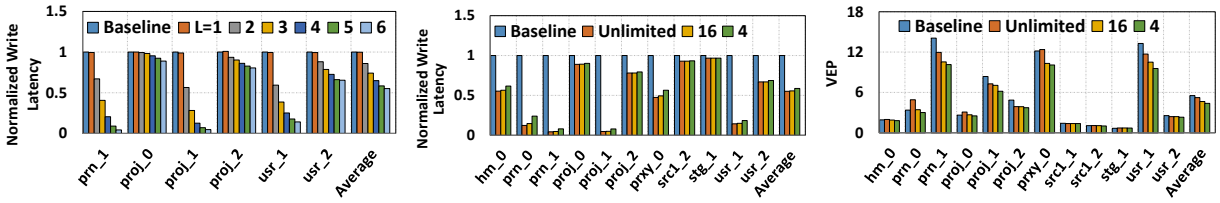
Write amplification: Figure 13a shows how our partial-erase enabled PEN system compared to the baseline system in terms of write-amplification. The write-amplification is reduced, on average, by 2.67x, compared to the baseline.

AEP and VEP: Figures 13b and 13c show the AEP and VEP, respectively. AEP improvements stem from the fewer erased pages during M-Merge, thanks to the partial-erase. On the contrary, VEP improvements result from the wear-leveling technique [30], the detailed sensitivity results of which can be found in Figure 14c.

GFTL comparison: Figure 12c plots the write latencies for the baseline and GFTL [9], *normalized* to our proposed PEN system. Note that GFTL is one kind of partial GC algorithm, which needs to reserve extreme long time for the block merge overhead to guarantee the constant request response time in the "Big Block". The figure shows that the GC algorithms designed for 2D NAND *cannot* directly be applied to the 3D NAND.

Sensitivity Results: Figure 14a plots the write latencies for our PEN system for different possible PB sizes (governed by L). As the possible number of PB sizes increases, PEN performs better, since a smaller PB reduces the copied valid pages in a PB during a block merge, ultimately reducing the overall block merge overheads.

Figures 14b and 14c plot the performance and reliability impact of the wear-leveling parameter, W, which is defined as the number of M-merge operations that can be performed on a block since the last baseline Merge for the block. A higher W value can provide better performance; but, it can also cause the severe skewness on erase count per page, which can be observed for prn_1, proj_0, and prxy_0 workload. A lower W value addresses the unevenness issue; but, it reduces the magnitude of performance improvement. As can be observed, W value 16 results in optimal performance and reliability. Figure 15a shows the write latency with and without considering the boundary program disturbance by the partial-erase operation. Our program-disturbance aware M-merge slightly increases the average write latency even under the most severe scenario where the NAND cells can only tolerate one partial-erase from the

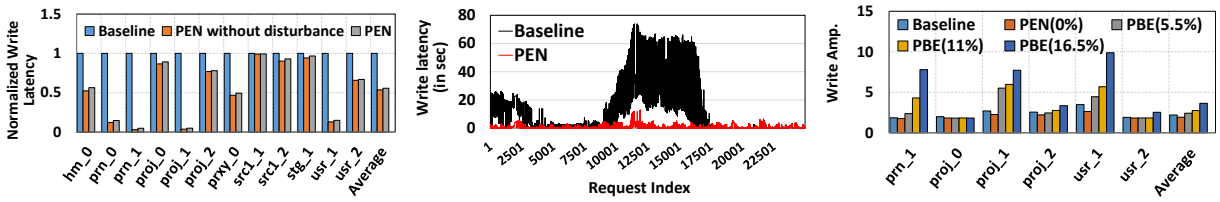


(a) Normalized write latency with different number of PB sizes for NFTL.

(b) Normalized write latency with different wear leveling parameters for NFTL.

(c) VEP with different wear leveling parameters for NFTL.

Figure 14: Sensitivity results in the case of NFTL.



(a) Normalized write latency with/without disturbance for NFTL.

(b) proj_1 write request time graph for NFTL.

(c) Write amplification comparison with [11].

Figure 15: Other aspects in the case of NFTL.

neighbor cells.

Variation of write latency over time: Figure 15b plots the write access latencies incurred by various I/O requests as they are processed in one of our workloads (proj_1). This graph clearly demonstrates that almost all the I/O write requests[5] experience reduced write access latency as they do *not* incur long stall times in the I/O Queue in our PEN system, unlike the baseline. As a result, we observe improvements in throughput.

Comparison with partial block erase (PBE) in [11]: Figure 15c compares PBE and PEN system. PBE enables the partial-erase by employing additional erase circuits between partial-blocks. Such implementation can mitigate the program-disturbance problem, but reduces the total available capacity. The detailed hardware and FTL modifications are not provided; as a result, we modeled the PBE system as a reduced capacity PEN system without program disturbance. The effect of loss in capacity by PBE can be observed in the reduced capacity change in the step from 0% to 16.5%. As a result, it incurs more GCs, and thus, PBE increases write amplification by 88% compared to PEN.

6.2.2 Hybrid FTL

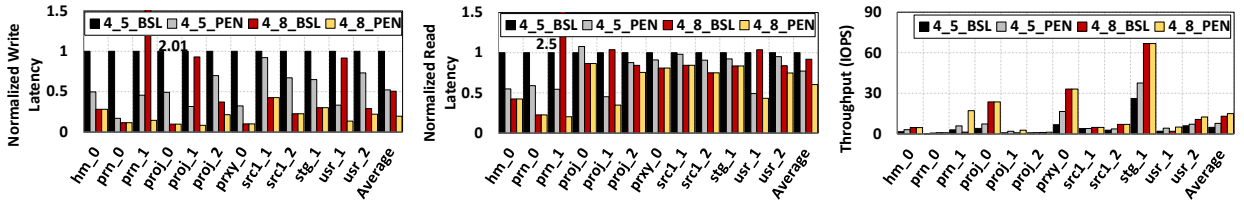
We now quantify the benefits of PEN over the baseline intra-/inter-superblock GC for Superblock FTL. We present the results for two different configurations, where the first one uses 4 logical blocks and 5 (physical) blocks, while the second one uses 4 logical blocks and 8 (physical) blocks. Figures 16a and 16b plot the normalized (with respect to the baseline) write and read latencies for the two configurations mentioned before. Workloads like proj_1 and usr_1 experience improved read/write latencies and enhanced throughput, due to the same reason explained in Section 6.2.1. In contrast, other workloads, especially hm_0 and stg_1, incur fewer inter-superblock GC; however, they incur more frequent intra-superblock GCs. Since the intra-superblock GC algorithm chooses the block in the superblock with the minimum number of valid pages as the victim block, not many valid pages need to be copied in the baseline. Hence, the avenues to improve the access latencies in PEN are minimized. Therefore, the cases in which PEN can outperform the intra-superblock GC are when the number of (physical) blocks is close to the number of logical blocks. Note that only prn_1 workload shows performance degradation in the baseline with more physical blocks, since the GC operations in the latter case are dominated by inter-superblock GC. Our PEN also reduces the write-amplification, AEP and VEP (Figures 17a, 17b, and 17c).

[5] We saw similar results in other workloads as well; but, we could not present them due to space constraints.

7 Related Work

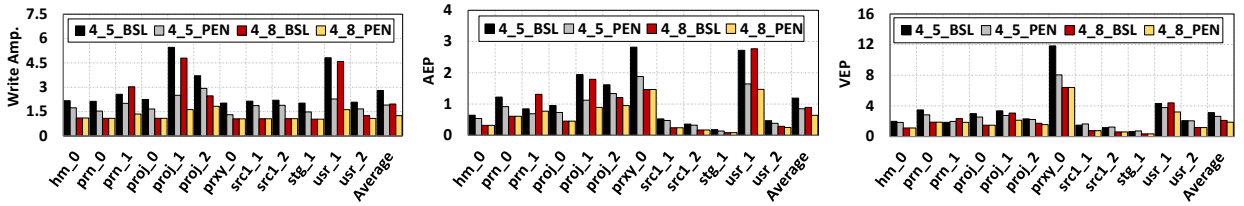
Partial-Erase proposals: Partial-Erase operation has been proposed [28] for 2D NAND flash, however, it did not pave way in to the actual products as it did not improve performance or reliability for 2D NAND flash. This is because the "Big block" problem is not as severe in 2D NAND compared to 3D NAND flash.

Partial-Erase operations for 3D NAND have recently been proposed in [11, 14]. However, due to the implementation diversity of 3D NAND, there are multiple ap-



(a) Normalized write latency. (b) Normalized read latency. (c) Throughput.

Figure 16: Performance improvements in the case of Superblock FTL. (BSL=Baseline)



(a) Write amplification. (b) AEP. (c) VEP.

Figure 17: Write amplification, AEP, and VEP improvements in the case of Superblock FTL. (BSL=Baseline)

proaches to enable such an operation. Partial block erase (PBE) [11] is discussed and compared against PEN in Section 6.2.1. Another implementation, subblock management [14] only allows three subblocks (two kinds of PB sizes) in a block. Such an implementation cannot provide any significant performance improvement, which can be observed in Figure 14a.

Besides the hardware-based partial-erase proposals, there also exist software-based proposals. Kim [30] proposes the strategy to address the wear-leveling problem caused by the partial-erase operation. Subblock erase [8] proposes a page-level FTL modification for the partial-erase operation in [11] to alleviate the "Big Block" problem. However, this proposal assumes that multiple partial-erase operations can be executed simultaneously, which necessitates non-trivial modifications to the underlying peripheral circuit components of the current NAND chips. In comparison, our PEN necessitates modest changes to the current peripheral circuitry. More importantly, the approach in [8] is only applicable to page-level FTL, which, as discussed before in Section 2.2.1, will become *impractical* in 3D NAND. In comparison, our approach focuses on the basic building block of GC, that is, the merge operation.

Partial GC proposals: We now compare our proposal to the "partial GC" research [7, 9, 23, 25] , conducted in the context of 2D NAND flash. Chang et al. [7] and Choudhari et al. [9] proposed periodic partial GC operations for real-time systems so that they provide minimal performance guarantees. GFTL [9] is discussed and compared in Section 6.2.1 and Figure 12c, respectively. AGCDGC and HIOS [23, 25] divide and distribute GC into more free-time slots, considering the address mapping and I/O request queue information, so that an SSD can have a more stable performance. Since these partial GC algorithms require additional knowledge to estimate free-time slots, they are FTL-specific and are not generic unlike our proposal. In addition, these partial GC algorithms still use coarse "block-level" erase operations, resulting in unnecessary valid pages copies during a GC operation, unlike our PEN.

8 Conclusion

In this paper, we propose and evaluate a novel partial-erase based PEN architecture in emerging 3D NAND flashes, which minimizes the number of valid pages copied during a GC operation. To show the effectiveness of our proposed partial-erase operation, we introduce our M-Merge algorithm that employs our partial-erase operation for NFTL and Superblock FTL. Our extensive experimental evaluations show that the average write latency under the proposed PEN system is reduced by 44.3% – 47.9%, compared to the baseline.

9 Acknowledgement

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). Kandemir and Jung are the co-corresponding authors. The authors thank Prof. Youjip Won for shepherding this paper.

References

[1] Micron mt29f8g08baa datasheet. https://www.micron.com/products/nand-flash/, Feb. 2007.

[2] AGARWAL, R., AND MARROW, M. A closed-form expression for write amplification in NAND flash. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE* (2010), pp. 1846–1850.

[3] BAN, A. Flash file system. https://www.google.com/patents/US5404485, Apr. 4 1995. US Patent 5,404,485.

[4] BOOTH, J. D., KOTRA, J. B., ZHAO, H., KANDEMIR, M., AND RAGHAVAN, P. Phase detection with hidden markov models for dvfs on many-core processors. In *2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS)* (2015).

[5] CAI, Y., MUTLU, O., HARATSCH, E. F., AND MAI, K. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *2013 IEEE 31st International Conference on Computer Design (ICCD)* (2013), IEEE.

[6] CAULFIELD, A. M., AND SWANSON, S. Quicksan: A storage area network for fast, distributed, solid state disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (2013).

[7] CHANG, L.-P., KUO, T.-W., AND LO, S.-W. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. Embed. Comput. Syst.* (Nov. 2004).

[8] CHEN, T.-Y., CHANG, Y.-H., HO, C.-C., AND CHEN, S.-H. Enabling sub-blocks erase management to boost the performance of 3d NAND flash memory. In *Proceedings of the 53rd Annual Design Automation Conference* (2016), DAC '16.

[9] CHOUDHURI, S., AND GIVARGIS, T. Deterministic service guarantees for nand flash using partial block cleaning. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (2008), CODES+ISSS '08.

[10] CO., S. E. Samsung V-NAND. http://www.samsung.com/semiconductor/products/flash-storage/v-nand/, 2016.

[11] D'ABREU, M. A. Partial block erase for a three dimensional (3d) memory. https://www.google.tl/patents/US9286989, May 19 2015. US Patent 9,286,989.

[12] DESNOYERS, P. Analytic Models of SSD Write Performance. *ACM Transactions on Storage*, 2 (Mar. 2014), 1–25.

[13] DIRIK, C., AND JACOB, B. The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (2009), ISCA '09.

[14] EUN CHU OH, J. K. Nonvolatile memory device and sub-block managing method thereof. https://www.google.com/patents/US20140063938, Mar. 6 2014. US Patent 2014/063938.

[15] GUPTA, A., KIM, Y., AND URGAONKAR, B. 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* (2009).

[16] HU, X.-Y., ELEFTHERIOU, E., HAAS, R., ILIADIS, I., AND PLETKA, R. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (2009), p. 10.

[17] HU, Y., JIANG, H., FENG, D., TIAN, L., LUO, H., AND ZHANG, S. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the international conference on Supercomputing* (2011).

[18] HUANG, J., BADAM, A., QURESHI, M. K., AND SCHWAN, K. Unified address translation for memory-mapped ssds with flashmap. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture* (2015).

[19] IM, J.-W., JEONG, W.-P., KIM, D.-H., NAM, S.-W., SHIM, D.-K., CHOI, M.-H., YOON, H.-J., KIM, D.-H., KIM, Y.-S., PARK, H.-W., AND OTHERS. 7.2 A 128gb 3b/cell V-NAND flash memory with 1gb/s I/O rate. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers* (2015), IEEE.

[20] JANG, J., KIM, H. S., CHO, W., CHO, H., KIM, J., SHIM, S. I., YOUNGGOAN, JEONG, J. H., SON, B. K., KIM, D. W., KIHYUN, SHIM, J. J., LIM, J. S., KIM, K. H., YI, S. Y., LIM, J. Y., CHUNG, D., MOON, H. C., HWANG, S., LEE, J. W., SON, Y. H., CHUNG, U. I., AND LEE, W. S. Vertical cell array using tcat(terabit cell array transistor) technology for ultra high density nand flash memory. In *2009 Symposium on VLSI Technology* (June 2009).

[21] JUNG, D., KANG, J.-U., JO, H., KIM, J.-S., AND LEE, J. Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme. *ACM Transactions on Embedded Computing Systems* (Mar. 2010).

[22] JUNG, M. Exploring parallel data access methods in emerging non-volatile memory systems. *IEEE Transactions on Parallel and Distributed Systems 28*, 3 (March 2017), 746–759.

[23] JUNG, M., CHOI, W., SRIKANTAIAH, S., YOO, J., AND KANDEMIR, M. T. Hios: A host interface i/o scheduler for solid state disks. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (2014).

[24] JUNG, M., AND KANDEMIR, M. Revisiting widely held ssd expectations and rethinking system-level implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (2013), SIGMETRICS '13.

[25] JUNG, M., PRABHAKAR, R., AND KANDEMIR, M. T. Taking garbage collection overheads off the critical path in ssds. In *Proceedings of the 13th International Middleware Conference* (2012).

[26] KANG, D., JEONG, W., KIM, C., KIM, D. H., CHO, Y. S., KANG, K. T., RYU, J., KANG, K. M., LEE, S., KIM, W., LEE, H., YU, J., CHOI, N., JANG, D. S., IHM, J. D., KIM, D., MIN, Y. S., KIM, M. S., PARK, A. S., SON, J. I., KIM, I. M., KWAK, P., JUNG, B. K., LEE, D. S., KIM, H., YANG, H. J., BYEON, D. S., PARK, K. T., KYUNG, K. H., AND CHOI, J. H. 7.1 256gb 3b/cell V-NAND flash memory with 48 stacked WL layers. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)* (Jan. 2016).

[27] KATSUMATA, R., KITO, M., FUKUZUMI, Y., KIDO, M., TANAKA, H., KOMORI, Y., ISHIDUKI, M., MATSUNAMI, J.,

FUJIWARA, T., NAGATA, Y., ZHANG, L., IWATA, Y., KIRISAWA, R., AOCHI, H., AND NITAYAMA, A. Pipe-shaped bics flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices. In *2009 Symposium on VLSI Technology* (June 2009).

[28] KI KIM, J. Partial block erase architecture for flash memory. `https://www.google.com/patents/US7804718`, Sept. 28 2010. US Patent 7,804,718.

[29] KIM, C., CHO, J. H., JEONG, W., PARK, I. H., PARK, H. W., KIM, D. H., KANG, D., LEE, S., LEE, J. S., KIM, W., PARK, J., AHN, Y. L., LEE, J., LEE, J. H., KIM, S., YOON, H. J., YU, J., CHOI, N., KWON, Y., KIM, N., JANG, H., PARK, J., SONG, S., PARK, Y., BANG, J., HONG, S., JEONG, B., KIM, H. J., LEE, C., MIN, Y. S., LEE, I., KIM, I. M., KIM, S. H., YOON, D., KIM, K. S., CHOI, Y., KIM, M., KIM, H., KWAK, P., IHM, J. D., BYEON, D. S., LEE, J. Y., PARK, K. T., AND KYUNG, K. H. 11.4 A 512gb 3b/cell 64-stacked WL 3d V-NAND flash memory. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (Feb. 2017).

[30] KIM, S.-H. Erasing method of non-volatile memory device. `https://www.google.com/patents/US9025389`, May 5 2015. US Patent 9,025,389.

[31] KIM, W., CHOI, S., SUNG, J., LEE, T., PARK, C., KO, H., JUNG, J., YOO, I., AND PARK, Y. Multi-layered vertical gate nand flash overcoming stacking limit for terabit density storage. In *2009 Symposium on VLSI Technology* (June 2009).

[32] KISLAL, O., , KANDEMIR, M. T., AND KOTRA, J. Cache-aware approximate computing for decision tree learning. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016).

[33] KOTRA, J. B., ARJOMAND, M., GUTTMAN, D., KANDEMIR, M. T., AND DAS, C. R. Re-NUCA: A practical nuca architecture for reram based last-level caches. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016).

[34] KOTRA, J. B., GUTTMAN, D., CHIDAMBARAM, N., AND KANDEMIR, M. T. Quantifying the potential benefits of on-chip near datacomputing in manycore processors. In *25th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (2017).

[35] KOTRA, J. B., KIM, S., MADDURI, K., AND KANDEMIR, M. T. Congestion-aware memory management on numa platforms: A vmware esxi case study. In *IEEE International Symposium on Workload Characterization (IIWSC)* (2017).

[36] KOTRA, J. B., SHAHIDI, N., CHISHTI, Z. A., AND KANDEMIR, M. T. Hardware-software co-design to mitigate dram refresh overheads: A case for refresh-aware process scheduling. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2017).

[37] KWON, M., ZHANG, J., PARK, G., CHOI, W., DONOFRIO, D., SHALF, J., KANDEMIR, M., AND JUNG, M. Tracetracker: Hardware/software co-evaluation for large-scale i/o workload reconstruction.

[38] LEE, S., SHIN, D., KIM, Y.-J., AND KIM, J. Last: Locality-aware sector translation for nand flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.* (Oct. 2008).

[39] LEE, S.-W., MOON, B., PARK, C., KIM, J.-M., AND KIM, S.-W. A case for flash memory ssd in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (2008).

[40] LEE, S.-W., PARK, D.-J., CHUNG, T.-S., LEE, D.-H., PARK, S., AND SONG, H.-J. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.* (July 2007).

[41] LI, C., SHILANE, P., DOUGLIS, F., SHIM, H., SMALDONE, S., AND WALLACE, G. Nitro: A capacity-optimized ssd cache for primary storage. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (June 2014), USENIX Association.

[42] LI, Y., LEE, P. P., LUI, J. C., AND XU, Y. Impact of Data Locality on Garbage Collection in SSDs: A General Analytical Study. pp. 305–315.

[43] LIU, J., KOTRA, J., DING, W., AND KANDEMIR, M. Network footprint reduction through data access and computation placement in noc-based manycores. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)* (2015).

[44] LUOJIE, X., AND KURKOSKI, B. M. An improved analytic expression for write amplification in NAND flash. In *Computing, Networking and Communications (ICNC), 2012 International Conference on* (2012), pp. 497–501.

[45] PARK, C., LEE, S., WON, Y., AND AHN, S. Practical implication of analytical models for ssd write amplification. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (2017), ICPE '17, pp. 257–262.

[46] PRINCE, B. *Vertical 3D Memory Technologies*. John Wiley & Sons, Inc., 2014.

[47] RINO MICHELONI, LUCA CRIPPA, A. M. *Inside NAND Flash Memory*. Springer Netherlands, 2010.

[48] SWAMINATHAN, K., KOTRA, J., LIU, H., SAMPSON, J., KANDEMIR, M., AND NARAYANAN, V. Thermal-aware application scheduling on device-heterogeneous embedded architectures. In *2015 28th International Conference on VLSI Design* (2015).

[49] SYNOPSYS. Hsim simulation reference manual. `http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSIM/Pages/default.aspx`, 2016. Synopsys HSIM.

[50] SYNOPSYS. Hspice. `https://www.synopsys.com/tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx`, 2016. Synopsys HSpice.

[51] TANAKA, H., KIDO, M., YAHASHI, K., OOMURA, M., KATSUMATA, R., KITO, M., FUKUZUMI, Y., SATO, M., NAGATA, Y., MATSUOKA, Y., IWATA, Y., AOCHI, H., AND NITAYAMA, A. Bit cost scalable technology with punch and plug process for ultra high density flash memory. In *2007 IEEE Symposium on VLSI Technology* (June 2007).

[52] TANG, X., KANDEMIR, M., YEDLAPALLI, P., AND KOTRA, J. Improving bank-level parallelism for irregular applications. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2016).

[53] YAMASHITA, R., MAGIA, S., HIGUCHI, T., YONEYA, K., YAMAMURA, T., MIZUKOSHI, H., ZAITSU, S., YAMASHITA, M., TOYAMA, S., KAMAE, N., LEE, J., CHEN, S., TAO, J., MAK, W., ZHANG, X., YU, Y., UTSUNOMIYA, Y., KATO, Y., SAKAI, M., MATSUMOTO, M., CHIBVONGODZE, H.,

OOKUMA, N., YABE, H., TAIGOR, S., SAMINENI, R., KODAMA, T., KAMATA, Y., NAMAI, Y., HUYNH, J., WANG, S. E., HE, Y., PHAM, T., SARAF, V., PETKAR, A., WATANABE, M., HAYASHI, K., SWARNKAR, P., MIWA, H., PRADHAN, A., DEY, S., DWIBEDY, D., XAVIER, T., BALAGA, M., AGARWAL, S., KULKARNI, S., PAPASAHEB, Z., DEORA, S., HONG, P., WEI, M., BALAKRISHNAN, G., ARIKI, T., VERMA, K., SIAU, C., DONG, Y., LU, C. H., MIWA, T., AND MOOGAT, F. 11.1 A 512gb 3b/cell flash memory on 64-word-line-layer BiCS technology. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (Feb. 2017).

[54] YANG, M. C., CHANG, Y. M., TSAO, C. W., HUANG, P. C., CHANG, Y. H., AND KUO, T. W. Garbage collection and wear leveling for flash memory: Past and future. In *Smart Computing (SMARTCOMP), 2014 International Conference on* (Nov 2014).

[55] YANG, Q., AND REN, J. I-cash: Intelligently coupled array of ssd and hdd. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture* (Feb 2011).

[56] YEDLAPALLI, P., KOTRA, J., KULTURSAY, E., KANDEMIR, M., DAS, C. R., AND SIVASUBRAMANIAM, A. Meeting midway: Improving cmp performance with memory-side prefetching. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2013).

[57] YOO, J., WON, Y., KANG, S., CHOI, J., YOON, S., AND CHA, J. Analytical model of ssd parallelism. In *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)* (Aug 2014), pp. 551–559.

[58] ZHANG, J., PARK, G., SHIHAB, M. M., DONOFRIO, D., SHALF, J., AND JUNG, M. Opennvm: An open-sourced fpga-based nvm controller for low level memory characterization. In *2015 33rd IEEE International Conference on Computer Design (ICCD)* (Oct 2015), pp. 666–673.