# Exploiting Process Similarity of 3D Flash Memory for High Performance SSDs

Youngseop Shim*, Myungsuk Kim*, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim
Department of Computer Science and Engineering, Seoul National University
{ysshim, morssola75, mjchun, jspark, yoonakim, jihong}@davinci.snu.ac.kr

## ABSTRACT

3D NAND flash memory exhibits two contrasting process characteristics from its manufacturing process. While process variability between different horizontal layers are well known, little has been systematically investigated about strong process similarity (PS) within the horizontal layer. In this paper, based on an extensive characterization study using real 3D flash chips, we show that 3D NAND flash memory possesses very strong process similarity within a 3D flash block: the word lines (WLs) on the same horizontal layer of the 3D flash block exhibit virtually equivalent reliability characteristics. This strong process similarity, which was not previously utilized, opens simple but effective new optimization opportunities for 3D flash memory. In this paper, we focus on exploiting the process similarity for improving the I/O latency. By carefully reusing various flash operating parameters monitored from accessing the leading WL, the remaining WLs on the same horizontal layer can be quickly accessed, avoiding unnecessary redundant steps for subsequent program and read operations. We also propose a new program sequence, called mixed order scheme (MOS), for 3D NAND flash memory which can further reduce the program latency. We have implemented a PS-aware FTL, called `cubeFTL`, which takes advantage of the proposed techniques. Our evaluation results show that `cubeFTL` can improve the IOPS by up to 48% over an existing PS-unaware FTL.

## CCS CONCEPTS

• **Computer systems organization** → *Firmware*; • **Hardware** → *Non-volatile memory*.

## KEYWORDS

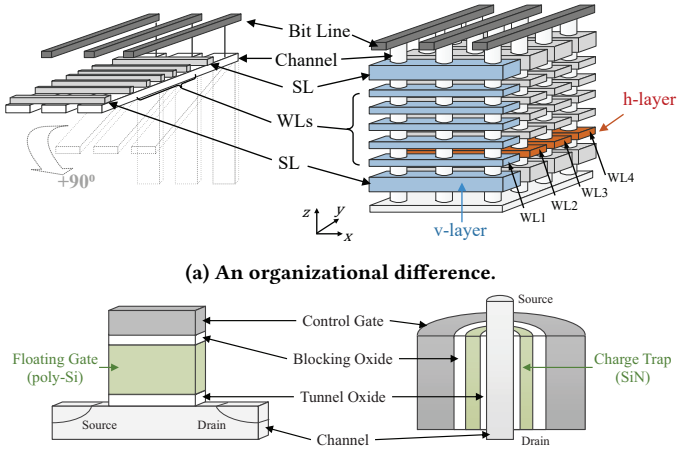
3D NAND Flash Memory, Process Similarity, Process Variability, SSD.

**ACM Reference Format:**
Youngseop Shim, Myungsuk Kim, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim. 2019. Exploiting Process Similarity of 3D Flash Memory for High Performance SSDs. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3352460.3358311

* Both authors contributed equally to this work.


*MICRO-52, October 12–16, 2019, Columbus, OH, USA*

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358311




**(a) An organizational difference.**

**(b) A difference in cell structures.**

**Figure 1: Illustrations of differences between 2D NAND and 3D NAND.**

## 1 INTRODUCTION

3D NAND flash memory [11, 14, 15, 32, 37], in which memory cells are vertically stacked, enabled the continuous growth in the flash capacity by overcoming various technical challenges in scaling 2D NAND flash memory. For example, 2D flash memory technologies had encountered the fundamental limits to scaling below the 10-nm process technology [34] because of the low device reliability (due to severe cell-to-cell interference) and high manufacturing complexity. By exploiting the vertical dimension for the capacity increase, 3D NAND flash memory has contributed to sustain the 50% per year growth rate in the NAND flash capacity [19], which established flash-based storage systems as de facto standards from mobile systems to high-performance enterprise environments.

Although an architecture of 3D NAND flash memory is conceptually described as if multiple 2D NAND flash layers are stacked in a vertical direction [17], the inner organization of 3D NAND flash memory is quite different from this logical explanation. Fig. 1(a) illustrates an organizational difference in a NAND block between 2D flash and 3D flash. The 3D NAND block in Fig. 1(a) consists of five horizontal layers (h-layers), which are stacked along the z axis. Each horizontal layer consists of four word lines (WLs). Similarly, the 3D NAND block may be described to have four vertical layers (v-layers) in the y axis where each v-layer consists of five vertically stacked WLs that are separated by select-line (SL) transistors. As shown in Fig. 1(a), when a 2D NAND block is rotated by 90 ° in a

counterclockwise direction using the x axis as an axis of rotation, it corresponds to a single v-layer. Furthermore, most 3D NAND devices (e.g., TCAT [15], p-BICs [18] and SMArT [7]) adopt cylindrical charge trap (CT)-type cell structures. This CT-type cell uses a non-conductive layer of silicon nitride (SiN) that traps electrical charges to store bit information, while 2D NAND devices use floating gate cell structures which store bits in a conductor (e.g., poly-Si). As shown in Fig. 1(b), this SiN layer has been modified into a three dimensional form that wraps around the channel, acting as an insulator that holds charges.

Since the overall organization and basic cell structure of 3D NAND flash memory were greatly changed over those of 2D NAND flash memory, the physical and electrical characteristics of 3D NAND flash memory are also quite different from those of 2D NAND flash memory. For example, 3D NAND flash memory is more reliable because the cell-to-cell interference, which causes data corruption, is not an issue in the CT-type cell structure. Since the cell dimension can be affected by the vertical etching process (described in Section 2), the error characteristics of 3D NAND flash memory vary significantly over the cell's geometric location within its 3D organization. Furthermore, there are several new reliability concerns (such as the early charge loss [5]) that were not present in 2D NAND flash memory. Therefore, in order to take full advantage of 3D flash memory, we need to revisit various optimization issues of NAND flash memory by exploiting the 3D NAND-specific characteristics.

Although several groups [25, 40] have already attempted to understand various characteristics of 3D flash memory, most of these studies focus on introducing new problems rather than presenting *practical solutions* to the new problems. Since 3D flash memory adds new sources of variation (such as the vertical dimension) from its manufacturing process, process variability tends to increase over 2D flash memory. This, in turn, makes it difficult to devise a solution that works well over the process variations originated from different sources. In this paper, in order to better understand the reliability characteristics of 3D flash memory (so that more practical solutions can be devised), we performed a comprehensive process characterization study using state-of-the-art 3D TLC NAND flash chips. We tested more than 20,000 flash blocks which were evenly selected from different physical locations using 160 flash chips. More than 11 million pages were used from these blocks.

From our characterization study, we have confirmed that there exists strong process variability between different h-layers as other researchers (such as [25]) have similarly observed. Since 3D NAND flash memory is manufactured using a vertically *successive* etching process from the topmost h-layer to the bottom h-layer, the cell structure along the z axis varies, leading to significant layer-to-layer variation. (We call this process variability *the vertical variability* or *inter-layer variability*.) Although several researchers (e.g., [6, 13, 25, 39]) have exploited the vertical variability for improving flash reliability, our result suggests that such techniques will be difficult to work effectively in practice. Although the inter-layer variability is clearly observed, its variation pattern is not easily predictable when several sources of variation (e.g., the flash aging status or the physical location of the flash block) are inter-related. Without a reasonable prediction model for the inter-layer variability, the effectiveness of the existing techniques can be quite limited. Our observation strongly motivates a need for better schemes to exploit the vertical variability in practice.

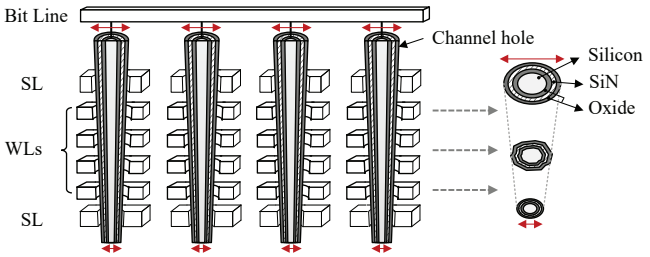
Our second finding from the process characterization study, which was a surprise for us, was that there exists very strong process similarity (PS) within a 3D flash block. Since the flash cells on the same h-layer of the 3D flash block experiences the same (etching) process conditions, the WLs on the same h-layer were expected to be quite *homogeneous* in their reliability characteristics. Our study, however, indicated a much stronger result than our expectation. That is, the WLs on the same h-layer exhibited virtually *equivalent* reliability characteristics. (We call this intra-layer process similarity *the horizontal similarity* or *intra-layer similarity*.) The horizontal similarity, which has not been previously reported in literatures, opens up simple but effective new optimization opportunities for 3D flash memory.

In this paper, we focus on exploiting the horizontal similarity for designing high-performance SSDs. Our key insight is the horizontal similarity can be effectively used during run time in deciding whether the required steps for implementing flash program and read operations can be safely skipped without degrading data reliability. Since the error characteristics of the WLs within the same h-layer are virtually equivalent, if we can obtain key parameters during accessing the leading WL of the h-layer, the remaining WLs of the same h-layer can be accessed efficiently by skipping unnecessary steps. By exploiting the horizontal similarity, we propose novel latency optimization techniques for program and read operations. Since the proposed latency optimization techniques are based on the parameter values monitored during run time, the vertical variability of 3D flash memory can be fully utilized as well. Unlike the existing techniques that depend on *off-line* parameter estimation [13, 25], our proposed techniques can accurately reflect changing environments by measuring key parameters *just-in-time* before they are used.

Since the effectiveness of the proposed optimization techniques depends on the NAND parameters monitored for the leading WL of each h-layer, we propose a modified program sequence for 3D NAND flash memory based on our characterization study. The modified program sequence, which we call the mixed order scheme (MOS), allows WLs in a single v-layer to be successively programmed. The extra flexibility of the MOS scheme in the page programming order comes with no sacrifice in the data reliability because little interference exists between different v-layers in 3D flash memory.

In order to evaluate the effectiveness of our proposed optimization techniques, we have implemented the PS-aware FTL, called `cubeFTL`, which fully exploits the intra-layer similarity of 3D flash memory using the emulated 3D flash memory which accurately reflects the cubic organization of 3D NAND flash memory. Our experimental results using various workloads show that `cubeFTL` can improve the IOPS by up to 48% over the existing PS-unaware FTL.

The rest of this paper is organized as follow. Before our optimization techniques are presented, we review the manufacturing process of 3D NAND flash memory and explain the steps of program and read operations in Section 2. In Section 3, we report our key findings from the characterization study. We describe the proposed PS-aware optimization techniques in Section 4 and PS-aware FTL is presented in Section 5. Experimental results follow in Section 6,




(a) A cross-sectional view. (b) A top-down view.

**Figure 2: A detailed illustration of a vertical layer of 3D NAND flash memory.**

and related work is summarized in Section 7. Section 8 concludes with a summary and future work.

## 2 BACKGROUND

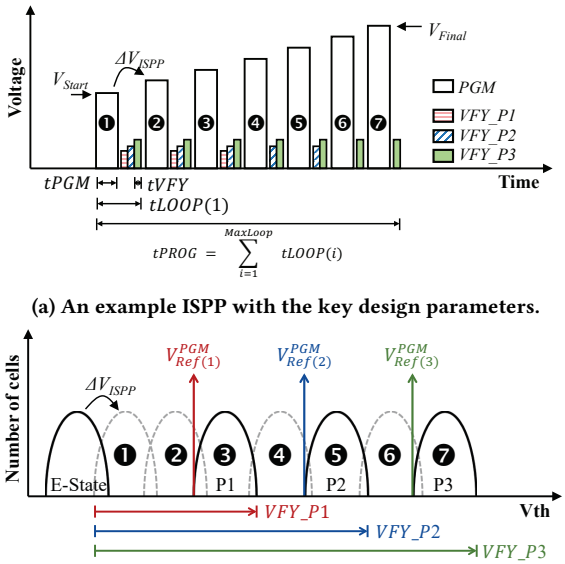
In order to optimize the latency of a program and read operations, our proposed techniques intelligently identify redundant steps using the horizontal similarity. In this section, we briefly give an overview of the 3D NAND manufacturing process that is directly related to the process characteristics reported in Section 3. We also review the basics of program operations and read operations at the micro-operation level where our proposed techniques are described.

### 2.1 3D NAND Manufacturing Process

Fig. 2 shows a detailed organization of a vertical layer in 3D NAND flash memory using a cross-sectional view (along the z axis) and a top-down view (of three cross sections along the x-y plane). The stacked cells are vertically connected through cylindrical channel holes. The channel holes are formed at the early stage of 3D NAND flash manufacturing by an etching process [15]. The etching process, which is considered as one of the most important steps for manufacturing 3D NAND flash memory, is the root cause of process similarity and variability in 3D NAND flash memory. While the etching process proceeds from the top h-layer to the bottom substrate, it introduces structural variations to channel holes in different h-layers. These structural variations, in turn, cause differences in flash cells' characteristics.

The inter-layer variability is the direct consequence of the structural variations from a high aspect ratio of channel holes. For example, as shown in Fig. 2(b), the diameter of the channel holes varies significantly over the height of an h-layer. The channel hole diameter in the topmost h-layer is wider than that in the bottom h-layer because of the high aspect ratio of the cylindrical channel hole. Furthermore, the shape of channel holes in some h-layers is an ellipse or a rugged shape unlike the circle in upper h-layers mainly due to etchant fluid dynamics.

Different channel hole diameters as well as their shapes can cause latency variations in the program or erase operation. Since a channel hole can be seen as a physical barrier to the charge flow along WLs in h-layers, the fluctuations in the channel hole sizes can increase the parasitic resistance or capacitance, thus resulting in large variations in the error characteristics of different h-layers. Since the electrical stress during NAND operations varies depending on the cell structure, the NAND aging status (e.g., endurance and data retention) can be also affected by the structural variations during the etching process.




(a) An example ISPP with the key design parameters.




(b) An example ISPP with changing needs for VFYs.

**Figure 3: An overview of the 2-bit MLC NAND program.**

Unlike the inter-layer variability between different h-layers, NAND cells on the same h-layer are *almost homogeneous* in their characteristics. Since these cells are manufactured by the same etching step at the same time, there is a little variation in the diameter of channel holes or in their shapes. This homogeneous nature of the etching step directly contributes to the intra-layer similarity.

### 2.2 NAND Flash Program Operation

A NAND flash memory chip consists of a large number of blocks, each of which is composed of multiple WLs. Depending on the number of bits per cell, one to four logical pages can be mapped to a single WL. For example, in TLC NAND flash memory, three logical pages are mapped into a single WL. When $s$ logical pages are mapped to a WL, each cell of the WL stores $s$ bits per cell by using $2^s$ different $V_{th}$ states. For example, MLC NAND flash memory stores two bits in a cell by using four different $V_{th}$ states.

In order to program a NAND cell to a specific state, the $V_{th}$ distribution should be formed within a specified $V_{th}$ window for the state. Flash devices generally use the incremental step pulse programming (ISPP) scheme [36] for controlling the $V_{th}$ distribution. Fig. 3(a) illustrates the ISPP scheme for programming 2-bit MLC NAND flash memory. The ISPP scheme gradually increases a program voltage by $\Delta V_{ISPP}$ until all cells in a page are positioned in their desired $V_{th}$ windows. After each program (shown by the white box in Fig. 3(a)), NAND cells are checked (i.e., *verified*) by a verify operation (shown by the shaded box in Fig. 3(a)) to see if they have been correctly programmed. We denote the program step and verify step by PGM and VFY, respectively. (Similarly, we use $tPGM$ and $tVFY$ to indicate the latency of PGM and VFY, respectively.) Once the cells were properly programmed, the verified cells are excluded (i.e., inhibited) from subsequent program steps. Otherwise, another round of the program-verify step is repeated. (Following

the convention, we call each sequence of program-verify steps by an ISPP loop.)

As shown in Fig. 3(a), a verify step is required after each program step unless the cells were already verified. For example, when cells are programmed for the P1 state, all the cells on a WL need to be verified including cells to be programmed, for example, for the P3 state. This is because the program speed of NAND cells on the same WL is *not the same*. Fast cells on a WL reach their target $V_{th}$ region with a small number of ISPP loops while a large number of ISPP loops are needed for slow cells on the same WL. In order to prevent the fast cells from exceeding the desired $V_{th}$ region, which causes the over-programmed errors, all programmed cells need VFY operations each time one ISPP loop is completed. As shown in Fig. 3(a), *tPROG* is defined as the total time of performing `MaxLoop` program-verify loops where the *i*-th loop takes *tLOOP(i)*. `MaxLoop` is set to ($V_{Final}$ - $V_{Start}$) / $\Delta V_{ISPP}$ where $V_{Final}$ and $V_{Start}$ are the ending program voltage and the starting program voltage, respectively. For the *i*-th ISPP loop, *tLOOP(i)* is given by the sum of *tPGM* and ($k_i \times tVFY$) where $k_i$ is the number of verify operations required for the *i*-th ISPP loop. In summary, *tPROG* can be expressed as follows:

$$tPROG = \sum_{i=1}^{\texttt{MaxLoop}} (tPGM + k_i \times tVFY) \quad (1)$$

Fig. 3(b) illustrates how $k_i$ values change over programmed states for 2-bit MLC flash. For P1-programmed cells, three ISPP loops (i.e., ❶, ❷, and ❸) are used and three VFY operations are needed (as shown in Fig. 3(a)) for each ISPP loop. (That is, $k_1 = k_2 = k_3 = 3$.) As shown in Fig. 3(b), although NAND cells are programmed for the P1 state, all the cells are needed to be verified against the P2 and P3 states as well because the program speeds of NAND cells on the same WL can be quite different. For P2-programmed cells, two more ISPP loops (i.e., ❹ and ❺) are needed with two VFY operations for each ISPP loop. (That is, $k_4 = k_5 = 2$.) Since the P1-programmed cells are excluded from ISPP loops for P2-programmed cells, each ISPP loop requires only two VFY operations, one for P2-programmed cells and the other for P3-programmed cells. Similarly, P3-programmed cells require two more ISPP loops (i.e., ❻ and ❼) with only one VFY operation in each ISPP loop. (That is, $k_6 = k_7 = 1$).

## 2.3 Read Retries in Read Latency

Since the data stored in flash memory cells can be corrupted by various error sources (such as a large number of P/E cycles or long data retention times [2, 4, 28]), when a NAND page with *m* different program states is read, an Error Correcting Code (ECC) engine in a flash controller monitors if the page includes errors beyond the ECC engine's correction capability. When such uncorrectable errors are detected, the flash controller retries the page read operation with a different combination of *m* $V_{Ref(i)}^{Read}$'s until the page does not include any uncorrectable errors. Note that $V_{Ref(i)}^{Read}$ is the read reference voltage used to distinguish *Pi* from *P*($i-1$). For example, in Fig. 4, $V_{Ref(3)}^{Read}$ distinguishes P3 from P2. When P3 is shifted due to a long data retention time, it may overlap with $V_{Ref(3)}^{Read}$, thus introducing bit errors [3, 22]. The shaded regions indicate such overlapped errors. When such a read failure occurs, $V_{Ref(i)}^{Read}$ is adjusted with

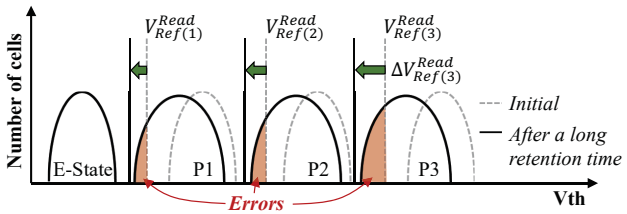



**Figure 4: An example of finding the optimal read reference voltages using read retries.**

an offset $\Delta V_{Ref(i)}^{Read}$. Since the read latency *tREAD* linearly increases as the number of read retries (`NumRetry`) increases, many optimization techniques such as [3] and [43] have been proposed for reducing `NumRetry`, in particular, when NAND cells reach near the end of their lifetimes. The core of these techniques relies on how quickly find the right offsets $\Delta V_{Ref(i)}^{Read}$'s. For example, the technique proposed in [43] keeps track of changes in $V_{th}$ distributions so that the optimal $\Delta V_{Ref(i)}^{Read}$'s can be quickly identified. Unlike the existing techniques, our proposed solution reduces `NumRetry` by exploiting the strong intra-layer similarity.

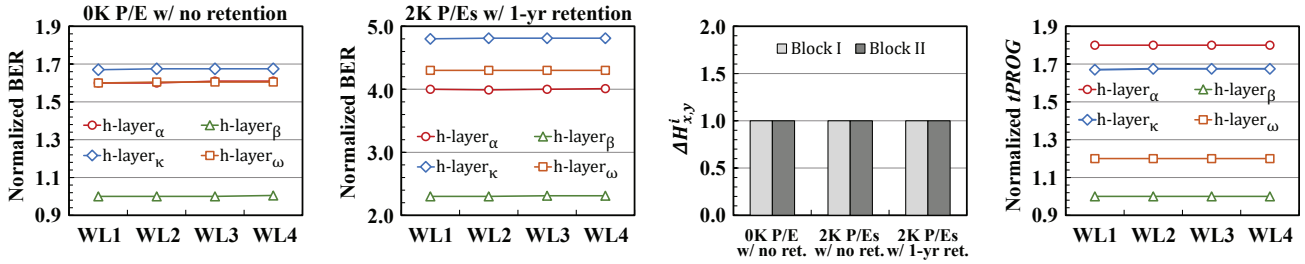
# 3 PROCESS VARIABILITY CHARACTERIZATION IN 3D NAND FLASH MEMORY

## 3.1 Variability Characterization Methodology

In order to better understand the implication of a cubic organization on the process variability of 3D NAND flash memory, we have performed a comprehensive process characterization study using 160 3D TLC flash chips with 48 horizontal layers where each layer consists of 4 WLs. We selected 128 blocks from each chip. A total of 11,520,000 pages (i.e., 3,840,000 WLs) were used in our study.

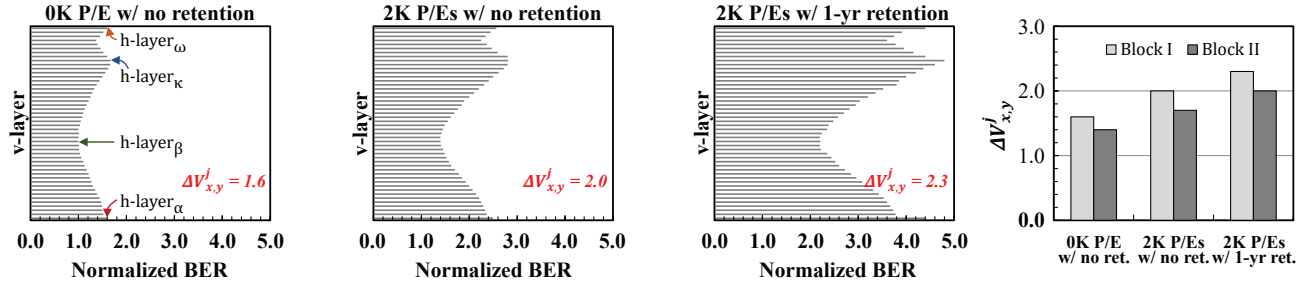
As a reliability measure of NAND memory cells, we used the number $N_{ret}(w_{ij}, x, t)$ of retention bit errors after *t*-month retention time when the WL $w_{ij}$ was *x* pre-cycled where $w_{ij}$ represents the *j*-th WL of the *i*-th h-layer in a flash block [16]. Since the main goal of our study was to understand the characteristics of process variability in 3D flash memory, we measured $N_{ret}(w_{ij}, x, t)$ values while changing both P/E cycles and data retention times using an in-house test board with a NAND controller and a temperature controller. For example, we varied P/E cycles from 0 (i.e., an initial condition) to 2K (i.e., the end of lifetime condition) and changed retention times from 0 month to 12 months.

In order to represent the degree of process variability among WLs, we define two metrics, $\Delta V_{x,t}^{j}$ and $\Delta H_{x,t}^{i}$, under a given ($x$, $t$). $\Delta V_{x,t}^{j}$ is defined as a ratio of the maximum $N_{ret}(w_{pj}, x, t)$ and minimum $N_{ret}(w_{qj}, x, t)$ where $w_{pj}$ and $w_{qj}$ are the WLs with the maximum number of retention bit errors and the minimum number of retention bit errors, respectively, along the *j*-th vertical layer. Similarly, $\Delta H_{x,t}^{i}$ is defined as a ratio of the maximum $N_{ret}(w_{ir}, x, t)$ and minimum $N_{ret}(w_{is}, x, t)$ where $w_{ir}$ and $w_{is}$ are the WLs with the maximum number of retention bit errors and the minimum number of retention bit errors, respectively, among the WLs in the *i*-th horizontal layer. $\Delta V_{x,t}^{j}$ indicates the degree of the inter-layer variability while $\Delta H_{x,t}^{i}$ shows the degree of the intra-layer variability. The larger $\Delta V_{x,t}^{j}$, the higher the process variability. On




(a) Normalized BERs. (b) Normalized BERs. (c) $\Delta H^i_{x,t}$ variations. (d) $tPROG$ comparisons.

**Figure 5: Characterization results on intra-layer similarity.**




(a) Normalized BERs. (b) Normalized BERs. (c) Normalized BERs. (d) $\Delta V^j_{x,t}$ variations.

**Figure 6: Characterization results on inter-layer variability.**

the other hand, as $\Delta V^j_{x,t}$ and $\Delta H^i_{x,t}$ get closer to 1, process similarity gets stronger.

## 3.2 Horizontal Intra-Layer Similarity

Fig. 5 summarizes the characterization results for intra-layer similarity. As shown in Figs. 5(a) and 5(b), no BER difference exists among four WLs on each h-layer shown. (In Figs. 5(a) and 5(b), we indicate four WLs on the same h-layer by WL1, WL2, WL3 and WL4.) We picked four representative h-layers whose vertical locations are shown in Fig. 6(a). The measured BERs were normalized over the h-layer which has the best retention BER. (BER is computed by dividing $N_{ret}(w_{ij}, x, t)$ by the total number of cells per WL.) As reported in other literatures (e.g., [33]), h-layer$_\alpha$ and h-layer$_\omega$, which are on the block edges, have high BER values. Although each h-layer shows different BER values because of channel-hole process variations, their $\Delta H^i_{x,t}$ values are all 1. In all of our measured h-layers, in fact, virtually all the $\Delta H^i_{x,t}$ values were 1 regardless of the flash aging conditions. Fig. 5(c) shows that the horizontal similarity is consistently maintained over different blocks as well under varying P/E cycles and retention times.

Strong horizontal intra-layer similarity implies that if we find out key parameters from *any* WL on the same h-layer, they can be safely applied to the remaining WLs on the same h-layer without the additional cost for estimating the parameters. For example, as shown in Fig. 5(d), all the WLs on the same h-layer have the same *tPROG*. If we measure *tPROG* once, say, for the leading WL on an h-layer, the *tPROG* values for the remaining WLs on the same h-layer can be exactly predicted. Since many flash optimization techniques (e.g., [1, 25]) rely on such parameter estimation for their high efficiency, the intra-layer similarity can be used in many different optimization cases in 3D flash memory. Furthermore, as the number of WLs on an h-layer is expected to increase in a near future (e.g., from 4 to 8 [21]), the efficiency of exploiting the intra-layer similarity will be further improved.

## 3.3 Vertical Inter-layer Variability

Fig. 6 summarizes the characterization results for inter-layer variability. Unlike the intra-layer similarity results shown in Figs. 5(a) and 5(b), significant h-layer to h-layer BER differences exist within tested flash blocks. (Because of the strong intra-layer similarity, we show BER values for the leading WL only in Figs. 6(a), 6(b) and 6(c).) All BER values were normalized over that of h-layer$_\beta$ (which is the most reliable h-layer) in a fresh NAND block (i.e., 0 P/E) with no retention time. Figs. 6(a), 6(b) and 6(c) also show that as WLs experience more P/E cycles, their BERs change in a complicated fashion so that it is very difficult to accurately predict the BER value of a WL under a given aging condition. For example, $\Delta V^j_{x,t}$ for the fresh block is only 1.6 while $\Delta V^j_{x,t}$ for the WLs after 1-year retention after 2K P/E cycles is 2.3. Furthermore, as shown in Fig. 6(c), when the flash block experiences a long retention time (e.g., 1-year retention time) around its maximum lifetime (i.e., 2K P/Es), BER values of less reliable h-layers (i.e. h-layer$_\kappa$, h-layer$_\alpha$, and h-layer$_\omega$) increase faster over those of more reliable h-layers (i.e., h-layer$_\beta$), introducing the nonlinear dynamic behavior. As shown in Fig. 6(d), there are significant per-block BER differences as well. In two sample blocks, Block I and Block II, $\Delta V^j_{x,t}$ of Block I is larger than that of Block II by 18%. Since the BER value of a WL can change in a nonlinear fashion under different aging conditions as well as the physical block location, it is a challenge to accurately estimate per-WL BER values, which are important for realizing the high efficiency of the existing techniques (such as [13, 25]) that exploit the inter-layer variability.

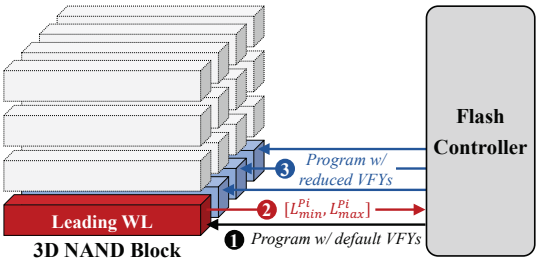



**Figure 7: An overall procedure for skipping redundant VFYs.**

# 4 PS-AWARE OPTIMIZATIONS

## 4.1 Program Latency Optimizations

In designing the optimized program scheme based on the horizontal similarity, we focus on two parts of Equation (1) for *tPROG*. First, we reduce $k_i$ by skipping redundant VFYs. (See Section 4.1.1.) Second, we reduce `MaxLoop`. (See Section 4.1.2.)

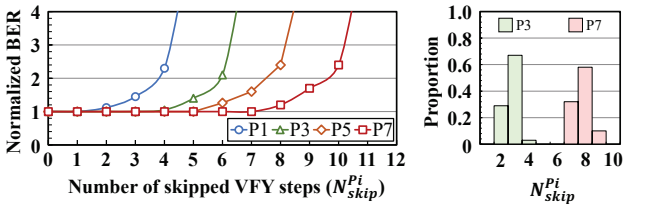
*4.1.1 Elimination of Redundant VFYs.* When a page is programmed by the ISPP scheme, if we knew the exact number of required ISPP loops for each cell *a priori*, no VFY would be necessary. Although this would be impossible in practice, thanks to the strong horizontal similarity, once the number of required VFY operations is known for the *leading* WL of an h-layer, we can accurately predict the number of ISPP loops for the subsequent WLs on the same h-layer, thus skipping unnecessary VFY steps.

In order to better describe our proposed technique, we rewrite Equation (1) as follows, assuming that there are *m* different program states, *P1, ..., Pm*:

$$tPROG = \sum_{s=P1}^{Pm} (L_s \times (tPGM + V_s \times tVFY)) \quad (2)$$

where $L_s$ is the number of ISPP loops required for *s*-programmed cells and $V_s$ is the number of VFY steps needed in a single ISPP loop for *s*-programmed cells. In $L_s$, we do not include earlier ISPP loops used for programming program states *P1, ..., P(s-1)*. For example, in Fig. 2(a), $L_{P1} = 3$, $L_{P2} = 2$, and $L_{P3} = 2$. Similarly, $V_{P1} = 3$, $V_{P2} = 2$, and $V_{P3} = 1$. If we knew $L_{P1}$, .., $L_{Pm}$ *in advance* before programming a WL, we could skip all the VFY steps (i.e., $\sum_{s=P1}^{Pm}(L_s \times V_s)$) from Equation 2.

In practice, however, because of significant program-speed variations among NAND cells in a WL, $L_{Pi}$ should be modelled as an interval $[L^{Pi}_{min}, L^{Pi}_{max}]$ rather than a constant. That is, for *Pi*-programmed cells, the smallest number of ISPP loops is $L^{Pi}_{min}$ while the largest number of ISPP loops is $L^{Pi}_{max}$. Since fast cells only need $L^{Pi}_{min}$ ISPP loops, VFY is necessary from the $L^{Pi}_{min}$-th ISPP loop so that verified cells are excluded from subsequent ISPP loops to avoid over-program errors. As shown in Fig. 7, in our VFY reduction technique, when cells in the leading WL of an h-layer are programmed to *Pi* (❶), we monitor both $L^{Pi}_{min}$ and $L^{Pi}_{max}$ (❷). Based on the strong horizontal similarity, when cells in the remaining WLs on the same h-layer are programmed to *Pi*, *N* VFY steps are skipped (❸) where $N = (\sum_{s=P1}^{P(i-1)} L^{S}_{max} + (L^{Pi}_{min} - 1))$ for $i > 1$ and $N = (L^{Pi}_{min} - 1)$ for $i = 1$. For example, if we assume that Fig. 3(a) in Section 2 shows the ISPP loops for the leading WL, in the remaining WLs, for P2-programmed cells, 3 VFYs are skipped (i.e., $L^{P1}_{max} + L^{P2}_{min} - 1$) while for P3-programmed cells, 5 VFYs are skipped (i.e., $L^{P1}_{max} + L^{P2}_{max} + L^{P3}_{min} - 1$ where $L^{P1}_{max} = 3$, $L^{P2}_{max} = 2$, $L^{P2}_{min} = 1$ and $L^{P3}_{min} = 1$).



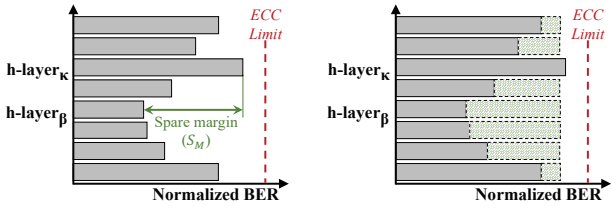

**(a) Relationship between $N^{Pi}_{skip}$ and BERs. (b) $N^{Pi}_{skip}$ distributions.**

**Figure 8: The effect of the number of skipped VFYs on the BER of program states.**

In our scheme, since higher states (e.g., P7 state in 3D TLC NAND) require more ISPP loops, they tend to skip more VFY steps over lower states (e.g., P3 state). Fig. 8 shows the typical effect of the number of skipped VFYs on the BER of program states assuming that $L^{Pi}_{min} = L^{Pi}_{max}$ for $i = 1, ..., m$. (All BER values normalized over the worst h-layer$_K$ at 2.0K P/Es with 1-year retention time.) P7-state cells can safely skip 7 VFYs while P1-state cells can skip only 1 VFY in TLC NAND flash memory. In all the program states, the more VFYs are skipped, the higher the BER value of the program states becomes. This is because when more VFYs are skipped, more fast cells get over-programmed. Skipped VFYs can reduce the average *tPROG* by 16.2% in our tested chips without degrading the flash reliability.

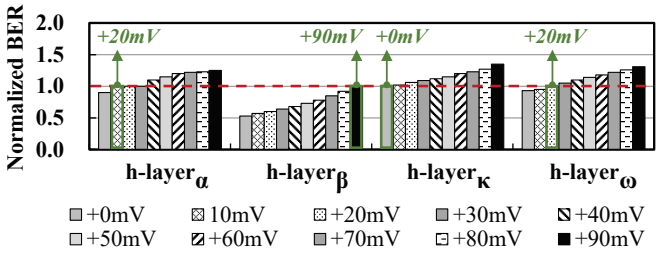
Although we assumed that $L^{Pi}_{min} = L^{Pi}_{max}$ in Fig. 8(a), in practice, they are not equal. Fig. 8(b) shows how the number of skipped VFYs may change depending on the program state by an example. For P7 state cells, $L^{Pi}_{min} = 7$ and $L^{Pi}_{max} = 9$. Since our technique measures $L^{Pi}_{min}$ and $L^{Pi}_{max}$ during run time, we can take full advantage of dynamic changes in skipping VFYs. On the other hand, if the number of skipped VFYs were statically determined off-line, many redundant VFYs may not be skipped because such techniques should use the smallest $L^{Pi}_{min}$ under the worst operating conditions.

*4.1.2 Reduction in Number of ISPP Loops.* Our second program latency optimization technique focuses on reducing `MaxLoop` in Equation (1) by *balancing* the BER of each h-layer without compromising the flash reliability. Our technique was mainly motivated by our observation that the default $V_{Start}$ and $V_{Final}$, which were set under the worst case reliability condition, can be safely relaxed by increasing $V_{Start}$ and decreasing $V_{Final}$ on most h-layers. With the modified $V'_{Start}$ and $V'_{Final}$ (where $V'_{Start} > V_{Start}$ and $V'_{Final} < V_{Final}$), `MaxLoop` becomes smaller because `MaxLoop` = $(V'_{Final} - V'_{Start}) / \Delta V_{ISPP}$. For example, if we can shorten the difference between $V_{Final}$ and $V_{Start}$ by 10%, `MaxLoop` is reduced by 10%, thus reducing the overall *tPROG* by 10%.

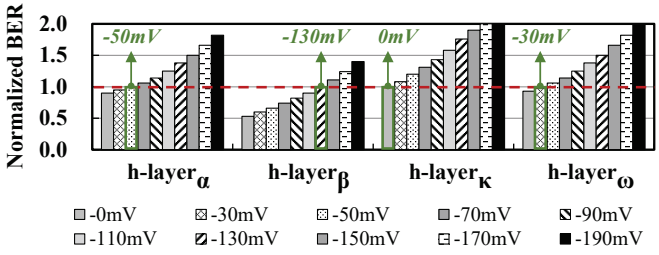



**(a) Before error balancing. (b) After error balancing.**

**Figure 9: The Error balancing between h-layers.**




(a) BER changes over $V_{Start}$ adjustment margins




(b) BER changes over $V_{Final}$ adjustment margins

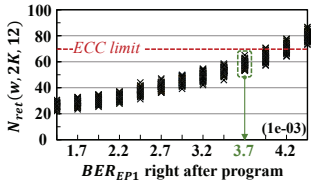
**Figure 10: $V_{Start}$ and $V_{Final}$ adjustment margins over different h-layers.**

Fig. 9 illustrates the key idea of our `MaxLoop` reduction technique by exploiting horizontal similarity. As shown in Fig. 9(a), in a default mode, $V_{Start}$ and $V_{Final}$ are set very conservatively to satisfy the data reliability at h-layer$_\kappa$ (i.e., the worst layer) under the worst operating conditions (e.g., at the end of the NAND lifetime with the longest data retention time). However, when WLs on most h-layers (e.g., h-layer$_\beta$) are programmed, considerable spare BER margins ($S_M$) exist because they are programmed under *non-worst* operating conditions and they are not on the *worst* h-layer. If $S_M$ were known in advance after programming the leading WL, by exploiting horizontal similarity, they could be effectively reused in adjusting $V_{Start}$ and $V_{Final}$ in subsequent WLs on the same h-layer, so that `MaxLoop` can be effectively reduced without any reliability problem. (Even with this adjustment of $V_{Start}$ and $V_{Final}$, as shown in Fig. 9(b), BER of h-layer$_\beta$ is still less than the ECC correction capability.)

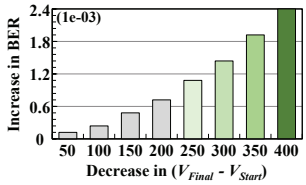
Although adjusting $V_{Start}$ and $V_{Final}$ adaptively by exploiting the inter-layer variability is not new (e.g., [13]), our scheme is novel in that $V_{Start}$ and $V_{Final}$ can be adjusted in a *tight but safe* fashion during run time. When the leading WL in the h-layer is programmed, we monitor the BER ($BER_{EP1}$) between the erase (E) state and the lowest program (P1) state.[1] $S_M$ is estimated as the difference between the monitored $BER_{EP1}$ and the maximum allowed BER ($BER_{EP1}^{Max}$). Since $S_M$ indicates how much BER can be relaxed without degrading the data reliability, it can be used as an efficient parameter in adjusting $V_{Start}$ and $V_{Final}$ for subsequent WLs on the same h-layer.

It should be noted that even for the same h-layer, $V_{Start}$ and $V_{Final}$ can be adjusted by different $S_M$ if operating conditions or the physical location of flash block change. This is the key advantage of our proposed scheme over the existing technique [13]. Unlike [13] which adjusts $V_{Final}$ based on the off-line characterization results

[1]It is known that the errors between E state and P1 state can accurately reflect the NAND health status and it can be used to predict the overall spare $V_{th}$ margin [20, 35].




(a) $BER_{EP1}$ vs. retention BERs.



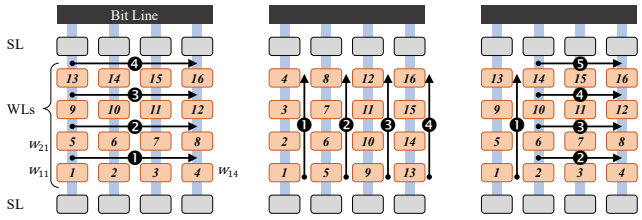

(b) ($V_{Final}$- $V_{Start}$) vs. BERs.

**Figure 11: $V_{Start}$ and $V_{Final}$ adjustment based on $BER_{EP1}$.**

(such as Fig. 10), our proposed scheme makes *on-line* decisions by adapting to changing operating conditions. In the scheme proposed in [13], for example, h-layer$_\beta$ in *all* flash blocks can decrease $V_{Final}$ by 130 mV only over its entire lifetime, resulting in an about 8% improvement in the program latency. On the other hand, our scheme not only relax the BER of h-layer by exploiting $S_M$ that reflects the difference originated from the flash aging and the position of the flash block, but also adjusts both $V_{Start}$ and $V_{Final}$, thus improving the program latency more efficiently.

In order to decide adjustment margins of $V_{Start}$ and $V_{Final}$ from the measured $BER_{EP1}$ of the leading WL of an h-layer, we first compute $S_M$ (= $BER_{EP1}^{Max} - BER_{EP1}$), which represents how much $BER_{EP1}$ can be relaxed over the maximum allowed $BER_{EP1}^{Max}$, shown in Fig. 11(a). From the computed $S_M$, we decide the total adjustment margin for $V_{Start}$ and $V_{Final}$. For example, as shown in Fig. 11(b), if the measured $BER_{EP1}$ indicates that $S_M$ is 1.7, the total adjustment margin for $V_{Start}$ and $V_{Final}$ is decided to be 320 mV. By adjusting $V_{Start}$ and $V_{Final}$ by 320 mV, $tPROG$ can be reduced by 19.7%. We experimentally decided $BER_{EP1}^{Max}$ from a large-scale characterization study. Similarly, a conversion table, which maps a given $S_M$ to a total adjustment margin for $V_{Start}$ and $V_{Final}$, is constructed off-line from extensive experimental measurements and used during run time. Once the total adjustment margin is known, we use another pre-defined table which states how to divide the total adjustment margin between $V_{Start}$ and $V_{Final}$.

*4.1.3 Modification in Program Sequence.* One interesting side effect of our proposed program latency optimization technique is that it divides WLs of a flash block into two distinct groups, leader WLs and follower WLs. The leader-WL group, which consists of leading WLs of each h-layer, has the default (i.e., normal) program latency. On the other hand, the follower-WL group, which includes all the remaining WLs except leader-WLs, has the reduced program latency. In the conventional program sequence shown in Fig. 12(a), which we call the horizontal-first order, as one leader-WL is programmed, three more WLs are added to the follower-WL group. Although the horizontal-first order scheme works well for many write workloads, when a large sequential write is required, its peak write bandwidth




(a) Horizontal-first order. (b) Vertical-first order. (c) Mixed order.

**Figure 12: Examples of different program orders in 3D flash.**

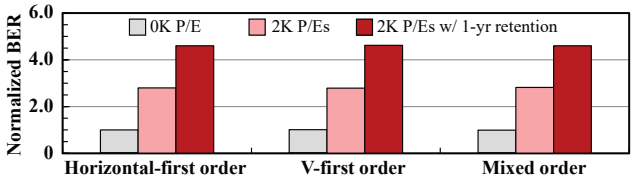



**Figure 13: Normalized BER comparisons over different program sequences.**

is limited because every four WL writes, one write should be served by the slow leader-WL group.

In order to increase the number of WLs in the faster follow-WL group, we have evaluated two additional program orders shown in Figs. 12(b) and 12(c), the vertical-first order scheme and the mixed order scheme (MOS), respectively. The key difference of these new order schemes over the conventional horizontal-first order is that they can reserve more WLs in the follower-WL group. For example, if we follow the mixed order scheme, all the WLs in the block except for ones in the first v-layer can be added to the follower-WL group. If the follower-WL group has a large number of WLs, write workloads with the high write bandwidth can be better served.

In order to validate the feasibility of our proposed program sequences, we evaluated the reliability characteristics of different program sequences using real 3D TLC flash chips. Fig. 13 shows the normalized BER results of three program sequences. Measured BER values were normalized over the BER value of the horizontal-first order. As shown in Fig. 13, three program sequences were virtually equivalent in their reliability.[2] Unlike 2D NAND flash, WLs in a 3D flash block are separated by SL transistors, so that each WLs can be programmed independently without interfering other WLs on the same h-layer. For example, when $w_{11}$ in Fig. 12 is programmed, other WLs ($w_{12}$, $w_{13}$, and $w_{14}$) are *inhibited* from being programmed by SL transistors [10], thus not introducing cell-to-cell interference as in 2D NAND flash.

*4.1.4 Safety Check for PS-aware Optimizations.* Our latency optimization techniques described in Sections 4.1.1 and 4.1.2 assume that the most recent $[L^{Pi}_{min}, L^{Pi}_{max}]$ values and $S_M$ are always applicable for subsequent program operations. Although this assumption is likely to hold on most practical settings, if there is a sudden change in NAND operating conditions (e.g., a sudden surge in the ambient temperature), the monitored $[L^{Pi}_{min}, L^{Pi}_{max}]$ and $BER_{EP1}$ values may be no longer valid to be used for optimizing *tPROG*. In order to avoid such rare failure cases, we check the BER value of each WL program operation. Since flash memory vendors provide the low-level NAND flash memory interface function, we can access the current settings of internal NAND flash memory (including modifications of operating parameters setting) and check the current BER value of each WL after program operation (e.g., Set/Get-Features) [30]. If the BER value of the just completed WL is significantly higher than that of the previously programmed WL on the same h-layer, we assume that the current WL was improperly programmed and re-program the same data in the following WL. When the same data is reprogrammed, new $[L^{Pi}_{min}, L^{Pi}_{max}]$ values and $S_M$ are monitored

[2]The maximum error difference among three sequences was less than 3%. This difference can be considered negligible because it comes mainly from the random telegraph noise (RTN) [27].

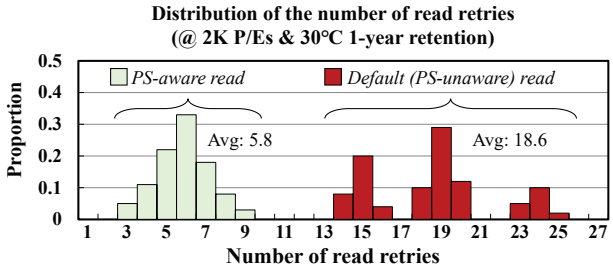



**Figure 14: Effect of the PS-aware read.**

for subsequent uses. Since we use the existing NAND interface with a minor code change, an additional time overhead is negligible (i.e., $< 1\ \mu s$) for supporting the extra safety check for our optimization techniques.

## 4.2 Read Latency Optimization

The read latency can be improved in a similar fashion as *tPROG* by exploiting the strong horizontal similarity characteristics of 3D NAND flash memory. As explained in Section 2, when the read operation fails, a flash controller retries the read operation with modified $V^{Read}_{Ref(i)}$'s until no uncorrectable error exists. To minimize the number of read retries, we aggressively reuse the most recent optimal $\Delta V^{Read}_{Ref(i)}$ offset values known for each h-layer. We denote the most recent offset values by $\mathbb{D} = \{\Delta V^{Read}_{Ref(i)} \mid 1 \le i \le m\}$ and assume that $\mathbb{D}$ was computed for a WL on an h-layer$_\mu$. If a requested read is to a WL on the same h-layer$_\mu$, we simply use $\mathbb{D}$ for the read request thanks to the strong horizontal similarity. Since each h-layer in a block has different $\mathbb{D}$ (as described in Section 3.3), when a read is to a WL not on the same h-layer$_\mu$, a new set $\mathbb{D}'$ is built.

Our proposed technique operates very accurately with low memory space overhead by exploiting strong horizontal similarity. Since the optimal $\Delta V^{Read}_{Ref(i)}$'s are rarely mispredicted[3] except when environmental factors (e.g., temperature) suddenly change, our proposed scheme effectively minimizes `NumRetry` without any hardware modification or any operational delay. Fig. 14 compares two distributions of `NumRetry` values under our scheme and the existing scheme, respectively, using real 3D TLC NAND flash chips. Our proposed technique can reduce `NumRetry` by 66% on average over the existing PS-unaware scheme, thus substantially improving *tREAD* of 3D flash memory.

# 5 CUBEFTL: PS-AWARE FTL

Based on PS-aware optimization techniques explained in Section 4, we have designed a novel FTL for 3D NAND flash-based SSDs, called `cubeFTL`. Fig. 15 describes an overall architecture of `cubeFTL`. As shown in Fig. 15, `cubeFTL` is based on a page-level FTL with two additional modules: *Optimal Parameter Manager (OPM)* and *WL Allocation Manager (WAM)*. The OPM plays a key role for PS-aware latency optimizations in `cubeFTL`. It sets optimal parameters for target WLs finish reads and programs with shorter latencies. The WAM is responsible for choosing WLs for every incoming host write. By considering the write performance asymmetry between leader WLs and follower ones, the WAM selects the most appropriate WL

[3]In case that the current optimal offset values fail to read, a new set $\mathbb{D}^*$ is computed and $\mathbb{D}^*$ is used for future prediction of optimal offset adjustments.

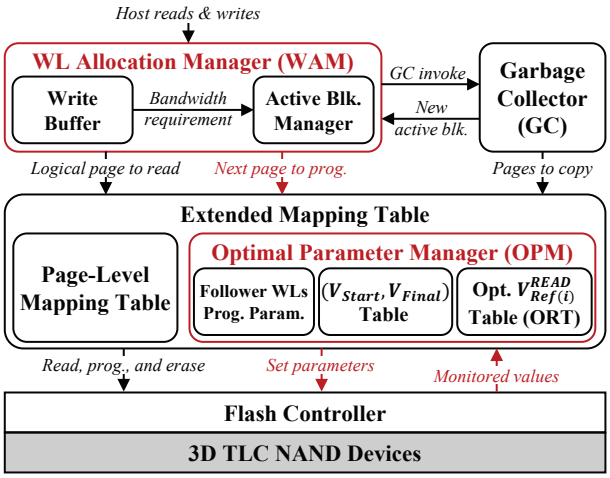



**Figure 15: An organizational overview of `cubeFTL`.**

under the current I/O workload characteristics. Once the WAM detects that the amount of free space is not sufficient, it invokes garbage collections (GCs).

## 5.1 Optimal Parameter Setting

The main role of the OPM is to set optimal parameters for a program and read operations based on PS-aware latency optimizations explained in Sections 4.1 and 4.2, respectively. In order to reduce *tPROG* for follower WLs [4], the OPM monitors two values from the leader WL program: 1) the number of ISPP loops for each program state (i.e., $[L^{Pi}_{min}, L^{Pi}_{max}]$) and 2) the BER value between the E state and P1 state (i.e., $BER_{EP1}$). Based on the first one, the OPM computes the number $N^{Pi}_{skip}$ of skipped VFYs for the *Pi*-programmed cells. VFY steps for each program states eliminates $N^{Pi}_{skip}$ VFY steps from the second WL. Using the monitored $BER_{EP1}$, the OPM obtains appropriate values of $V_{Start}$ and $V_{Final}$ referring a predefined table. The optimal program parameters, $N^{Pi}_{skip}$'s, $V_{Start}$ and $V_{Final}$ are temporarily kept until they are used for all the follower WLs.

For the read latency optimization, the OPM maintains an optimal read reference voltage table (ORT) to keep track of the most recent $\Delta V^{Read}_{Ref(i)}$ values for every h-layer in an SSD (i.e., $\mathbb{D}_h$, where $h$ is an h-layer index in an SSD). For reading a WL from an h-layer $h$, if the corresponding $\mathbb{D}_h$ in the ORT is different from the default value, the OPM adjusts $\Delta V^{Read}_{Ref(i)}$'s for the read as values of $\mathbb{D}_h$ to reduce `NumRetry`. If read retries are newly incurred (either by misprediction or additional retention), the OPM updates the ORT with the final $\Delta V^{Read}_{Ref(i)}$'s which did not introduce uncorrectable errors. The space overhead for maintaining the ORT is trivial. For example, with 3D TLC NAND devices, if there are 4 adjustable $\Delta V^{Read}_{Ref(i)}$ levels between each program state, two bytes per h-layer is sufficient for representing 7 (i.e., $2^3 - 1$) $\Delta V^{Read}_{Ref(i)}$'s in $\mathbb{D}_h$ ($7 \times 2 = 14$ bits). Therefore, when each h-layer has 4 WLs and the page size is 16 KB, the space overhead is about 0.001% ($2[\frac{byte}{h\text{-}layer}] \times (16,384[\frac{byte}{page}] \times 3[\frac{page}{WL}] \times 4[\frac{WL}{h\text{-}layer}])^{-1} \approx 1.02 \times 10^{-5}$), which requires only 10 MB memory for a 1-TB SSD.

[4] `CubeFTL` does not exploit the vertical variability (due to its inconsistency explained Section 3.3), so leader WLs are programmed with default parameters (i.e., no *tPROG* reduction for leader WLs).

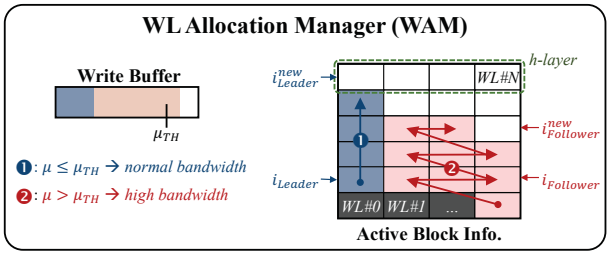



**Figure 16: Adaptive WL allocations by the OPM.**

Setting optimal parameters for each operation can be supported by using the existing NAND interface. Since most NAND manufacturers already provide the low-level interface for adjusting various parameters, no significant hardware modification is necessary for the PS-aware latency optimizations in `cubeFTL`. Note that setting parameters only takes less than 1 $\mu$s, therefore, the performance overhead is negligible considering the average *tPROG* (e.g., 700 $\mu$s) and *tREAD* (e.g., 80 $\mu$s).

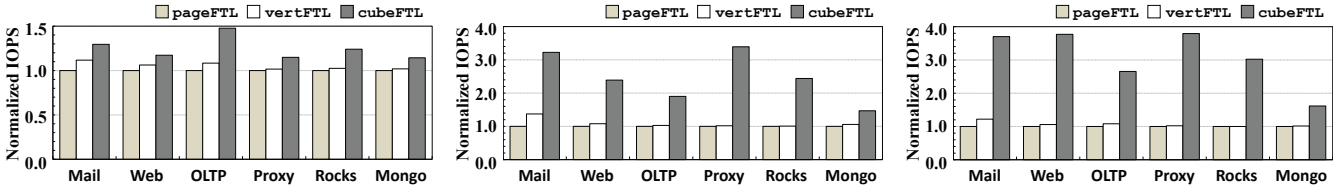
## 5.2 Adaptive WL Allocation

In order to take a full advantage of proposed techniques at the storage system level, the WAM selects the most appropriate WLs for each host write, taking into account the current performance requirement. As a general guideline, the WAM tries to handle as many host writes as possible with fast follower WLs under a high write-bandwidth requirement, while using slow leader WLs when the normal program speed is sufficient. By doing so, the WAM enables `cubeFTL` to better meet varying performance requirements.

The WAM monitors the write buffer utilization $\mu$ to estimate the current write-bandwidth requirement. For a given threshold value $\mu_{TH}$ (e.g., 0.9), the WAM judges that a high write-bandwidth is required if $\mu > \mu_{TH}$. Otherwise, it estimates that no high write-bandwidth is necessary. Depending on the estimated requirement, the WAM selects the right WL in active blocks (i.e., the current write point) for flushing next entry of the write buffer. Following the above guideline, if $\mu > \mu_{TH}$, the WAM tries to use follower WLs. Otherwise, it prefers slow leader WLs.

For such workload-aware WL allocations, the WAM should be able to flexibly choose a desired WL from a block. To this end, as shown in Fig. 16, it manages active blocks in a *fully* mixed fashion based on the MOS. The WAM keeps track of two h-layer indices for each active block; $i_{Leader}$ points to the h-layer with the next free leader WL while $i_{Follower}$ points to the h-layer with the next free follower WL. When $\mu \le \mu_{TH}$, the WAM flushes the write buffer by using leader WLs from $i_{Leader}$ (❶), even if follower WLs of lower h-layers were not yet used. On the other hand, under a high write-bandwidth requirement, follower WLs are successively used as long as $i_{Follower} < i_{Leader}$ (❷). This allows `cubeFTL` to use more follower WLs when the write buffer is heavily used so that more free buffer space can be made available quickly for the next host write request.

When no high write bandwidth is required, `cubeFTL` tries to use leader WLs. However, once `cubeFTL` runs out of leader WLs in an active block, `cubeFTL` must use follower WLs even when no high write bandwidth is needed. In order to avoid such an awkward situation, we use two active blocks per chip where more than two active blocks per chip could be better. However, the more active




(a) Normalized IOPS in 0K P/E w/ no ret. (b) Normalized IOPS in 2K P/Es w/ 1-month ret. (c) Normalized IOPS in 2K P/Es w/ 1-year ret.

**Figure 17: Comparisons of normalized IOPS under six different workloads.**

blocks per chip, the more memory overhead for the OPM to keep optimal parameters for follower WLs. In the current cubeFTL, the WAM manages two active blocks per chip as a simple solution.

# 6 EXPERIMENTAL RESULTS

## 6.1 Experimental Settings

In order to evaluate the effectiveness of the proposed techniques, we have implemented cubeFTL on a unified SSD development platform for NAND flash-based storage systems [23]. Our evaluation platform can support up to the 512-GB capacity, but for fast evaluation, the storage capacity was set to 32 GB. The target SSD was configured to have 2 buses, each of which has 4 3D TLC NAND chips. Each chip consists of 428 blocks and each block consists of 48 h-layers with 4 WLs per h-layer. The page size was set to 16 KB. For our evaluation, we modified a NAND flash model in our platform to distinguish leader WLs from follower WLs. For leader WLs, typical operating parameters were applied [19, 41]. For follower WLs, *tPROG* was shortened by up to 35.9% and NumRetry was reduced by 66% on average[5]. We have compared cubeFTL with two different FTLs: pageFTL and vertFTL. PageFTL, which is our baseline FTL, is a page-level mapping FTL without any 3D NAND-specific optimizations. VertFTL, which employs a MaxLoop reduction technique proposed in [13], is used to represent the existing state-of-the-art technique for reducing *tPROG*. VertFTL, however, adjusts $V_{Final}$ only in a conservative manner to ensure the flash reliability even under the worst operating condition.

For our evaluations, we used six workloads, four from the Filebench benchmark tool [38], Mail, Web, Proxy and OLTP, and two real-world DB applications, Rocks and Mongo. Mail, Web, and Proxy, emulate I/O activities of mail servers, web servers, and proxy cache servers, respectively, while OLTP represents intensive DB workloads. Rocks and Mongo run RocksDB [9] and MongoDB [29], respectively, using Yahoo! Cloud Service Benchmark (YCSB) [8]. Among many different YCSB workloads, we used update-heavy workload (type A) which consists of 50/50 reads and writes.

## 6.2 Performance Evaluation

In order to compare the overall performance of cubeFTL over pageFTL and vertFTL, we measured IOPS values of each FTL under different P/E cycles and retention times. Since NumRetry highly depends on the SSD aging condition, evaluation under different aging states are necessary to understand the effectiveness of the proposed read latency optimization. Three different aging states at 30°C are used in our evaluations: 0K P/E cycle with no retention (i.e., fresh state), 2K P/E cycles with 1-month retention, and 2K P/E cycles with 1-year retention. From our characterization results, partly shown in Fig. 14, we constructed a probabilistic read-retry model for our experiments. We assumed that no read retry occurs in the fresh state. After 2K P/E cycles, 30% and 90% of reads need read retries, respectively, with 1-month retention time and 1-year retention time.

Fig. 17(a) shows normalized IOPS values of three FTLs under the fresh state. Note that no read retries occurred in fresh blocks, so all the performance gains of cubeFTL came from the program latency optimization. As shown Fig. 17(a), cubeFTL significantly improves I/O performance over the other FTLs under every workload. It outperforms pageFTL and vertFTL by up to 48% and 36%, respectively. As expected, the performance gains of vertFTL over pageFTL was insignificant compared to cubeFTL, due to its conservative $V_{Final}$ adjustment. VertFTL can reduce *tPROG* by only 8% on average while the 30% *tPROG* reduction is possible in cubeFTL on average. For OLTP, cubeFTL achieved the largest IOPS improvement ratio. Since OLTP was the most write-intensive workload, cubeFTL took advantage of its workload-aware adaptive WL allocation heuristics of the WAM module. By allocating fast follower WLs when burst writes are requested, cubeFTL can improve the IOPS by 48% over pageFTL.

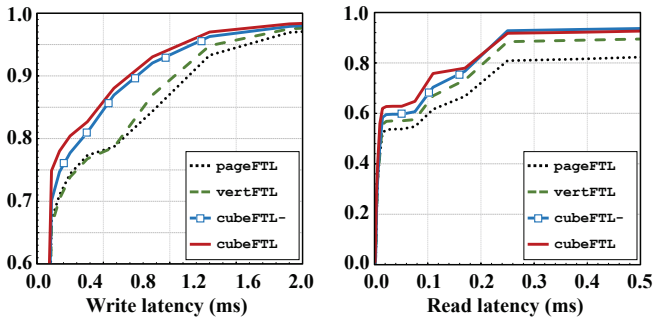
Figs. 17(b) and 17(c) show the normalized IOPS under 1-month retention time and 1-year retention time after 2K P/E cycles, respectively. As shown in Figs. 17(b) and 17(c), IOPS gains of cubeFTL over the other FTLs are significantly increased, compared to that of Fig. 17(a) when all the blocks were fresh. Unlike Fig. 17(a), the largest IOPS gain by cubeFTL was observed in Proxy, not OLTP. This is because, at the end of the SSD lifetime, the read retry overhead can dominate the overall performance. CubeFTL achieved a higher improvement ratio because of its efficient NumRetry reduction technique.

## 6.3 Impact of Adaptive WL Allocation

In order to evaluate the effectiveness of the workload-aware WL allocations in cubeFTL, we evaluated the I/O latency using Rocks workload under the fresh block state using cubeFTL−. CubeFTL− works in the same fashion as cubeFTL except that the WAM module of cubeFTL is disabled. CubeFTL− handles host writes without considering the current performance requirement, using WLs in an active block based on the horizontal-first order.

Fig. 18(a) shows a cumulative distribution of the write latency under the Rocks workload. As shown in Fig. 18(a), both the cubeFTL and cubeFTL− served write requests with short latencies over pageFTL and vertFTL. For example, in cubeFTL, the 90th percentile of the write latency was 0.72 ms. On the other hand, in pageFTL, the 90th percentile of the write latency was 1.10 ms,

[5]These parameter changes reflect the overall effect of our proposed techniques described in Secs. 4.1 and 4.2.




**(a) Cumulative distributions of the write latency.** **(b) Cumulative distributions of the read latency.**

**Figure 18: Comparisons of the I/O latency in Rocks.**

about 1.53 times longer. Compared to cubeFTL, cubeFTL− performed slightly lower. For example, the 80th percentile of cubeFTL was about 42.26% shorter than that of cubeFTL−. By using more follower WLs for burst writes, cubeFTL can flush the write buffer quickly, allowing more host writes to be returned early.

It is noteworthy that the adaptive WL allocation was also effective in improving the application-level read latency, which greatly affects the user-perceived performance. Fig. 18(b) shows a cumulative distribution of the read latency under the Rocks workload. Even though our proposed *tREAD* optimization has no impact at the fresh state (because no read retry occurs), cubeFTL reduced the read latency over the other FTLs including cubeFTL−. This is because that less reads were blocked by previous writes owing to the fast flush of the write buffer in cubeFTL. In conclusion, the WAM enabled cubeFTL to maximize the benefit of the PS-aware latency optimizations at the system level.

## 7 RELATED WORK

There have been intensive investigations to reduce the NAND flash memory program latency (such as [13, 16, 24, 26, 31, 39]) and read latency (such as [25]). However, most existing techniques do not exploit the intra-layer similarity of 3D NAND flash memory. For example, the technique proposed by Pan *et al.* [31], which targets for 2D NAND flash memory, reduces MaxLoop by dynamically increasing $\Delta V_{ISPP}$. Although this technique can improve *tPROG*, it requires an extra safety mechanism to handle pages written in an accelerated fashion. Since no process similarity or variability is considered, the efficiency of this technique is quite limited.

A few investigators have recently proposed 3D NAND-specific optimization techniques. However, none of these techniques exploit the process similarity of 3D NAND flash memory in a comprehensive fashion as our work. In order to reduce the program latency, for example, Hung. *et al.* [13] proposed a program latency improvement technique by exploiting inter-layer variability, but they do not consider dynamic characteristic changes of horizontal layers that occur when P/E cycles, retention times and physical block location change. Therefore, its practical benefit is rather limited. Furthermore, no horizontal similarity was exploited at all. Wang *et al.* [39] also proposed a reliability enhancement technique by exploiting the inter-layer variability, but they do not consider the horizontal similarity as our work. Maejima *et al.* [26] presented a MaxLoop reduction technique similar to ours using the horizontal similarity. However, their technique, whose details were not described, is limited to adjusting $V_{Start}$ only while our technique adjusts both $V_{Start}$ and $V_{Final}$. Furthermore, our proposed techniques treat the I/O latency optimization problem in a comprehensive fashion including the VFY reduction technique, the new program sequence and the reduction in read retries.

For the read latency improvement, Luo *et al.* [25] proposed a read reference voltage adjustment technique exploiting the vertical variability. However, they neither consider the dynamically changing operating conditions nor exploit the horizontal similarity. Since their technique depends on the static flash operating parameters that were decided off-line, its efficiency is rather limited by not taking advantage of additional inter-layer variability.

## 8 CONCLUSIONS

We have presented novel I/O latency optimization techniques for 3D flash-based SSDs. Our proposed techniques exploit the horizontal intra-layer similarity, which was newly discovered from our characterization study. Strong intra-layer similarity allows flash operating parameters monitored for the leading WL of an h-layer to be reused in a tight but safe fashion for the remaining WLs on the same h-layer. In order to reduce *tPROG*, when the leading WL of an h-layer is programmed, we monitor the number of required ISPP loops for each program state and the spare BER margin, and then reuse them for the remaining WLs on the same h-layer to skip redundant VFYs and reduce MaxLoop. We also propose a modified program order, called the mixed order, which can improve the peak write bandwidth when combined with the proposed program latency optimization techniques. For reducing *tREAD*, we reuse the optimal $\Delta V^{Read}_{Ref(i)}$ offset values from the first read from an h-layer so that following reads can reuse the optimal $\Delta V^{Read}_{Ref(i)}$ monitored from the first read. Our experimental results using cubeFTL show that the proposed techniques can improve IOPS by up to 48% over the existing PS-unaware FTL. The high efficiency of the proposed techniques comes from directly measuring key flash operating parameters during run time so that changing operating conditions are accurately reflected in the optimization procedure in a high adaptive fashion.

Although we have focused on improving the I/O latency of high-performance SSDs in this paper, we believe that the strong intra-layer similarity can be widely used for optimizing different aspects of high-performance SSDs. For example, it may be possible to improve the quality and speed of an error-correction coding algorithm used for such SSDs by exploiting various information collected from the leader WL. Since the horizontal similarity guarantees accurate I/O response times, it can be used to build SSDs with a highly deterministic latency as a solution to the long-tail problem in SSDs [12, 42].

## ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for the feedback and comments. This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1701-11. The ICT at Seoul National University provided research facilities for this study (*Corresponding Author: Jihong Kim*).

# REFERENCES


[1] Y. Cai, S. Ghose, E. Haratsch, Y. Luo, and O. Mutlu. 2017. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *IEEE Journal of Solid-State Circuits* 105, 9 (2017), 1666–1704.

[2] Y. Cai, E. Haratsch, O. Mutlu, and K. Mai. 2012. Error patterns in MLC NAND flash memory: measurement, characterization, and analysis. In *Proceedings of the Automation and Test in Europe (DATE)*.

[3] Y. Cai, Y. Luo, E. Haratsch, K. Mai, and O. Mutlu. 2015. Data retention in MLC NAND flash memory: characterization, optimization, and recovery. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[4] Y. Cai, O. Mutlu, E. Haratsch, and K. Mai. 2013. Program interference in MLC NAND flash memory: characterization, modeling, and mitigation. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*.

[5] C. Chen, H. Lue, C. Hsieh, K. Chang, K. Hsieh, and C. Lu. 2010. Study of fast initial charge loss and it's impact on the programmed states Vt distribution of charge-trapping NAND flash. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[6] S. Chen, Y. Chen, H. Wei, and W. Shih. 2017. Boosting the performance of 3D charge trap NAND flash with asymmetric feature process size characteristic. In *Proceedings of the Design Automation Conference (DAC)*.

[7] E. Choi and S. Park. 2012. Device considerations for high density and highly reliable 3D NAND flash cell in near future. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[8] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[9] Facebook. 2013. RocksDB. http://rocksdb.org/.

[10] K. Fukuda, Y. Shimizu, K. Amemiya, M. Kamoshida, and C. Hu. 2007. Random telegraph noise in flash memories-model and technology scaling. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[11] Y. Fukuzumi, Y. Matsuoka, M. Kito, M. Kido, M. Sato, H. Tanaka, Y. Nagata, Y.and Iwata, H. Aochi, and A. Nitayama. 2007. Optimal integration and characteristics of vertical array devices for ultra-high density, bit-cost scalable flash memory. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[12] M. Hao, G. Soundararajan, D. Kenchammana-Hosekote, A. Chien, and H. Gunawi. 2014. The Tail at Store: a revelation from millions of hours of disk and SSD deployments. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*.

[13] C. Hung, M. Chang, Y. Yang, Y. Kuo, T. Lai, S. Shen, J. Hsu, S. Hung, H. Lue, Y. Shih, S. Huang, T. Chen, T. Chen, C. Chen, C. Hung, and C. Lu. 2015. Layer-aware program-and-read schemes for 3D stackable vertical-gate BE-SONOS NAND flash against cross-layer process variations. *IEEE Journal of Solid-State Circuits* 50, 6 (2015), 1491–1501.

[14] M. Ishiduki, Y. Fukuzumi, R. Katsumata, M. Kito, M. Kido, H. Tanaka, Y. Komori, Y. Nagata, T. Fujiwara, T. Maeda, Y. Mikajiri, S. Oota, M. Honda, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama. 2009. Optimal device structure for pipe-shaped BiCS flash memory for ultra high density storage device with excellent performance and reliability. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[15] J. Jang, H. Kim, W. Cho, H. Cho, J. Kim, S. Shim, Y. Jang, J. Jeong, B. Son, D. Kim, K. Kim, J. Shim, J. Lim, K. Kim, S. Yi, J. Lim, D. Chung, H. Moon, S. Hwang, J. Lee, Y. Son, Y. Chung, and Y. Lee. 2009. Vertical cell array using TCAT (Terabit Cell Array Transistor) technology for ultra high density NAND flash memory. In *Proceedings of the IEEE Symposium on VLSI Technology (VLSI)*.

[16] J. Jeong, Y. Song, S. Hahn, S. Lee, and J. Kim. 2017. Dynamic erase voltage and time scaling for extending lifetime of NAND flash-based SSDs. *IEEE Trans. Comput.* 66, 4 (2017), 616–630.

[17] S. Jung, J. Jang, W. Cho, H. Cho, J. Jeong, Y. Chang, J. Kim, Y. Rah, Y. Son, J. Park, M. Song, K. Kim, J. Lim, and K. Kim. 2006. Three dimensionally stacked NAND flash memory technology using stacking single crystal Si layers on ILD and TANOS structure for beyond 30nm node. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*.

[18] R. Katsumata, M. Kito, Y. Fukuzumi, M. Kido, H. Tanaka, Y. Komori, M. Ishiduki, J. Matsunami, T. Fujiwara, Y. Nagata, L. Zhang, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama. 2009. Pipe-shaped BiCS flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices. In *Proceedings of the Symposium on VLSI Technology (VLSI)*.

[19] C. Kim, D. Kim, W. Jeong, H. Kim, I. Park, H. Park, J. Lee, J. Park, Y. Ahn, J. Lee, J. Lee, S. Kim, H. Yoon, J. Yu, N. Choi, Y. Kwon, N. Kim, H. Jang, J. Park, S. Song, Y. Park, J. Bang, S. Hong, B. Jeong, H. Kim, C. Lee, Y. Min, I. Lee, I. Kim, S. Kim, D. Yoon, K. Kim, Y. Choi, M. Kim, H. Kim, P. Kwak, J. Ihm, D. Byeon, J. Lee, K. Park, and K. Kyung. 2018. A 512Gb 3b/cell 64-stacked WL 3D-NAND flash memory. *IEEE Journal of Solid-State Circuits* 53, 1 (2018), 124–133.

[20] M. Kim, Y. Song, M. Jung, and J. Kim. 2018. SARO: a state-aware reliability optimization technique for high density NAND flash memory. In *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*.

[21] Mark Lapedus. 2018. 3D NAND flash wars begin. https://semiengineering.com/3d-nand-flash-wars-begin/.

[22] J. Lee, J. Choi, D. Park, and K. Kim. 2003. Data retention characteristics of sub-100 nm NAND flash memory cells. *IEEE Electron Device Letters* 24, 12 (2003), 748–750.

[23] S. Lee, J. Park, and J. Kim. 2012. FlashBench: a workbench for a rapid development of flash-based storage devices. In *Proceedings of the 23rd IEEE International Symposium on Rapid System Prototyping*.

[24] R. Liu, C. Yang, and W. Wu. 2012. Optimizing NAND flash-based SSDs via retention relaxation. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*.

[25] Y. Luo, S. Ghose, Y. Cai, E. Haratsch, and O. Mutlu. 2018. Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation. In *Proceedings of the ACM Measurement and Analysis of Computing Systems (POMACS)*.

[26] 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. 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 *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*.

[27] R. Micheloni. 2016. *3D Flash Memories*. Springer Netherlands.

[28] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill. 2008. Bit error rate in NAND flash memories. In *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*.

[29] MongoDB. 2009. MongoDB. https://www.mongodb.com/.

[30] ONFI Workgroup. 2017. Open NAND flash interface specification 4.1. http://www.onfi.org/~/media/onfi/specs/onfi_4_1_gold.pdf?la=en.

[31] Y. Pan, G. Dong, and T. Zhang. 2011. Exploiting memory device wear-out dynamics to improve NAND flash memory system performance. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*.

[32] K. Park, D. Byeon, and D. Kim. 2014. A world's first product of three-dimensional vertical NAND flash memory and beyond. In *Proceedings of the Non-Volatile Memory Technology Symposium (NVMTS)*.

[33] K. Park, S. Lee, J. Sel, J. Choi, and K. Kim. 2007. Scalable wordline shielding scheme using dummy cell beyond 40 nm NAND flash memory for eliminating abnormal disturb of edge memory cell. *Japanese Journal of Applied Physics* 46, 4 (2007), 2188–2192.

[34] Y. Park, J. Lee, S. Cho, G. Jin, and E. Jung. 2014. Scaling and reliability of NAND flash devices. In *Proceedings of the IEEE Symposium on Reliability Physics (IRPS)*.

[35] X. Shi, S. Wu, F. Wang, C. Xie, and Z. Lu. 2018. Program error rate-based wear leveling for NAND flash memory. In *Proceedings of the Automation and Test in Europe (DATE)*.

[36] K. Suh, B. Suh, Y. Lim, J. Kim, Y. Choi, Y. Koh, S. Lee, S. Kwon, B. Choi, J. Yum, J. Choi, J. Kim, and H. Lim. 1995. A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme. *IEEE Journal of Solid-State Circuits* 30, 11 (1995), 1149–1156.

[37] H. Tanaka, M. Kido, K. Yahashi, M. Oomura, R. Katsumata, M. Kito, Y. Fukuzumi, M. Sato, Y. Nagata, Y. Matsuoka, Y. Iwata, H Aochi, and A. Nitayama. 2007. Bit cost scalable technology with punch and plug process for ultra high density flash memory. In *Proceedings of the IEEE Symposium on VLSI Technology (VLSI)*.

[38] T. Vasily, Z. Erez, and S. Spencer. 2016. Filebench: a flexible framework for file system benchmarking. *;login: The USENIX Magazine* 41, 1 (2016), 6–12.

[39] Y. Wang, L. Dong, and R. Mao. 2017. P-Alloc: process-variation tolerant reliability management for 3D charge-trapping flash memory. *ACM Transactions on Embedded Computer Systems* 16, 5 (2017), 1–19.

[40] Y. Luo, S. Ghose, Y. Cai, E. Haratsch, and O. Mutlu. 2018. HeatWatch: improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[41] 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, U. 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. 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. Lu, T. Miwa, and F. Moogat. 2017. A 512Gb 3b/cell flash memory on 64-word-line-layer BiCS technology. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*.

[42] S. Yan, H. Li, M. Hao, M. Tong, S. Sundararaman, A. Chien, and H. Gunawi. 2017. Tiny-Tail Flash: near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *Proceedings of the USENIX Conference on File and Storage*

*Technologies (FAST).*

[43] J. Yang. 2011. High-efficiency SSD for reliable data storage systems. *Flash Memory Summit (FMS)* (2011).