ApproxFTL: On the Performance and Lifetime Improvement of 3-D NAND Flash-Based SSDs

Jinhua Cui, Youtao Zhang, *Member, IEEE*, Liang Shi, *Member, IEEE*, Chun Jason Xue, *Member, IEEE*, Weiguo Wu, and Jun Yang

***Abstract*—3-D NAND flash is one of the most prospective advances in flash memory industry. While 3-D flash improves cell density and reduces lithography cost through die stacking, it suffers from severe program disturbance, which leads to significant performance and lifetime degradation for 3-D flash-based SSDs. To address the above challenge, we propose ApproxFTL, an approximate-write aware flash translation layer design, that uses *approximate-write* operations to store error-resilient data of modern applications. By reducing the maximal threshold voltage and tightening the guard bands between multilevel cell states, approximate write operations not only finish early but also exhibit large disturbance reduction, which can be exploited to alleviate disturbance in physical blocks that save both precise and approximate data. ApproxFTL maximizes the disturbance mitigation through approximate-write aware data placement, wear leveling, and garbage collection enhancements. Our experimental results show that ApproxFTL, while preserving high data quality, improves the read and write response time of flash accesses by 41.38% and 45.64% on average, respectively, and extends the lifetime of 3-D flash-based SSDs by 5.75% when comparing to the state-of-the-art.**

***Index Terms*—3-D flash memory, approximate storage, reliability.**

Manuscript received July 7, 2017; revised October 14, 2017; accepted November 26, 2017. Date of publication December 13, 2017; date of current version September 18, 2018. This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1000303 and Grant 2016YFB0201800, in part by the Joint Research Fund for Overseas Chinese, Hong Kong and Macau Young Scientists of the National Natural Science Foundation of China under Grant 61628210, in part by the National Natural Science Foundation of China under Grant 91630206, Grant 61672423, and Grant 61772092, in part by the National Science Foundation of the United States under Grant CCF-1718080, and in part by the National 863 Program under Grant 2015AA015304. The work of J. Cui was supported by the Chinese Scholarship Council under Grant 201606280098. This paper was recommended by Associate Editor M.-F. Chang. *(Jinhua Cui and Youtao Zhang contributed equally to this work.) (Corresponding authors: Jinhua Cui; Youtao Zhang.)*

J. Cui and W. Wu are with the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: cjhnicole@gmail.com; wgwu@xjtu.edu.cn).

Y. Zhang is with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 USA (e-mail: zhangyt@cs.pitt.edu).

L. Shi is with the College of Computer Science, Chongqing University, Chongqing 440044, China, and also with the Key Laboratory of Cyber Physical Society Credible Service Computing, Ministry of Education, Chongqing 440044, China (e-mail: shi.liang.hk@gmail.com).

C. J. Xue is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: jasonxue@cityu.edu.hk).

J. Yang is with the Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: juy9@pitt.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2782765

I. Introduction

THREE-DIMENSIONAL (3-D) NAND flash is one of the most prospective advances in flash memory industry [1]. By stacking flash cells in vertical direction, 3-D flash achieves significant density improvement and lithography cost reduction. However, a 3-D flash page has more neighboring pages than a planar page does. As such, programming a 3-D flash page disturbs more pages in the same block[1] while a programmed page is subject to more disturbing write operations, i.e., when its neighboring pages are programmed. With fast technology scaling, program disturbance is projected to be one of the major challenges in 3-D flash [2].

Most schemes developed for mitigating program disturbance in 3-D flash were proposed at the hardware level, e.g., those that redesign the cell structures [3] and those that redesign the ECC circuit [4]. At the system level, Wang *et al.* [5] proposed to reorder writes sent to physical blocks to minimize interblock disturbance. Chang *et al.* [6] proposed to reorder writes to pages within one physical block. Chang *et al.* [7] proposed to mitigate disturbance by redesigning the program operation and sequence. Chang *et al.* [8] proposed to split a large 3-D block to sub-blocks with lower disturbance.

In this paper, we propose ApproxFTL, an approximate-write aware flash translation layer (FTL) to mitigate program disturbance in 3-D flash and address the performance and lifetime degradation of 3-D flash-based SSDs. A recent work that is close to our design is to trade data accuracy for improved solid state memory access performance [9]. By exploiting the error resilience in modern applications, Sampson *et al.* proposed to reduce the number of write steps when writing error resilient data. The technique was designed for phase change memory (PCM) but can be adapted to multilevel cell (MLC) flash. A major difference between their approach and ours is that the former cannot mitigate program disturbance, which is the main design goal of this paper. The following summarizes our contributions.

1) We propose to realize approximate flash write, i.e., page programming, by reducing the maximal threshold voltage and tightening the guard bands between MLC flash states. The approximate-write not only speeds up flash accesses to approximate data but also greatly reduces the disturbance to neighboring pages.

[1]In this paper, we focus on VG 3-D flash cell architecture that has one block spread across multiple layers. The design is applicable to other 3-D cell architectures.

0278-0070 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

2) We then propose ApproxFTL, an approximate-write aware FTL design, to fully exploit the benefits of approximate-write operations. It integrates three enhancements in FTL.
   a) We enhance the baseline data allocation to prioritize the checkerboard-style allocation of approximate and precise pages in physical blocks, which effectively minimizes the disturbance on precise pages in each block.
   b) We enhance the baseline wear leveling to exploit the lifetime benefit from reducing maximal threshold voltage. By tracking the wearing effect based on the data allocation pattern of different blocks, we distribute approximate and precise writes proportionally to each physical block.
   c) We enhance the baseline garbage collection (GC) such that, at the GC time, it moves valid approximate and precise pages in two batches to appropriate physical blocks for maximized disturbance reduction.
3) We evaluate the proposed schemes and compare them to the state-of-the-art. Our experimental results show that ApproxFTL, while preserving high data quality, improves the read and write response time of flash accesses by 41.38% and 45.64% on average, respectively, and extends the lifetime of 3-D flash-based SSDs by 5.75% when comparing to the state-of-the-art.

The rest of this paper is organized as follows. Section II presents the background and related work. Section III discusses the approximate page programming that mitigates program disturbance in 3-D flash. Section IV elaborates the design details of ApproxFTL. Section V lists the experiment settings and analyzes the results. We conclude this paper in Section VI.

II. Background and Related Work

A. 3-D NAND Flash Memory

3-D NAND flash is a type of flash that achieves density improvement through die stacking. A single etching can punch through multiple layers and connect the cells from these layers [1]. 3-D NAND flash memory can be classified into two types based on the flowing direction of the string current: 1) vertical channel (VC) and 2) vertical gate (VG). In VC 3-D flash, e.g., BiCS [10] and V-NAND [11], the current flows vertically while in VG 3-D flash, e.g., 3DVG [4], [12], the current flows horizontally so that the gate signals are vertically shared. Studies have shown that VG flash has better pitch scalability due to its smaller minimal cell size and well-controlled interface [6], [13]. In this paper, we elaborate our design using VG 3-D flash while the proposed schemes are applicable to VC 3-D flash.

Fig. 1 presents a block equivalent circuit schematic of two-layer 3-D flash. A physical block in 3-D flash consists of $N \times L$ pages, where L is the number of layers of the flash module and N is the number of pages in each layer that belong to the block. The pages from different layers are vertically stacked so that their corresponding wordlines (WLs) are spliced together along the Z direction. The layout within each layer is traditional, that is, a bitline (BL) has contacts with all WLs in one layer for the given block while the bits from one page are laid out along the WL direction.

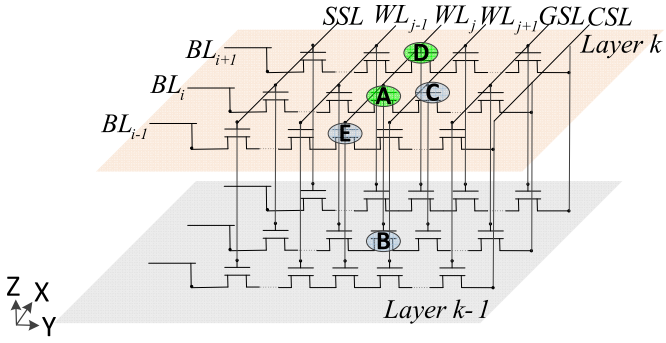


Fig. 1. Circuit schematic of two-layer VG 3-D NAND flash.

1) Program Disturbance to Be Mitigated in This Paper: As the number of layers is increasing in 3-D flash, the deteriorated program disturbance is an extremely critical design issue in 3-D flash memory [6], [7]. That is, the threshold voltage of a programmed flash page may be unintentionally shifted when programming its neighboring pages, which is mainly attributed to parasitic capacitances coupling effect between adjacent ones [7]. In the example, when programming cells A and D, we may disturb their neighboring cells, e.g., cells B and C. Studies have shown that this type of disturbance becomes much larger in 3-D flash than that in 2-D flash. Hsieh *et al.* [14] showed that the disturbance along Y direction is about 1.33 times of that along Z direction in the same chip [14]. In this paper, we focus on mitigating capacitive coupling effect-based program disturbance, which is the same as most of existing 3-D disturbance mitigation designs [6], [15], [16].

By adapting the threshold voltage shift models in planar flash [17], [18], the threshold voltage shift on the cell being disturbed in 3-D flash memory, due to the capacitive coupling effect-based program disturbance, can be modeled as following.

Given a cell s at (i, j, k) where i is cell offset within one flash page (along BL-to-BL, or X direction), j is the page index on one layer (along WL-to-WL, or Y direction), k is the layer index (along layer-to-layer, or Z direction). We have $0 \leq i <$ PSIZE, $0 \leq j < N$; $1 \leq k \leq L$, and PSIZE, N, and L are the total number of cells in one page, the number of pages in one layer, and the number of layers, respectively. For discussion clarity, we skip the cells in the top/bottom layers and at the outer lines of each block. They have fewer neighboring cells and are also handled in the experiments

$$\begin{aligned} \Delta V_{(s)} &= \sum_{\forall t} \frac{C_{(s)(t)} \quad \Delta V_{(t)}}{C_{\text{total}}} \\ s &= (i, j, k) \\ t &\in \{(i, j-1, k), (i, j+1, k) \\ &\qquad (i, j, k-1), (i, j, k+1)\} \end{aligned} \tag{1}$$

where $C_{\texttt{total}}$ is the total capacitance of the victim cell; $C_{(s)(t)}$ is the coupling capacitance between the victim cell s and the

its neighboring cell t. For example, cell $(i, j, k+1)$ is the one on top of the victim cell. The four neighboring cells along Y and Z directions have non-negligible disturbance while the disturbance effects along WL direction and diagonal directions are often small and neglected [14]. $\Delta V_{(t)}$ indicates the threshold voltage change before and after programming cell t. Cell s and t are referred to victim (or disturbed) and disturbing cell, respectively, in the rest of this paper.

A victim cell s gets disturbed if any of its neighboring cells t along Y and Z directions is programmed after programming s, according to (1). For a typical sequential programmed order that programs cells layer-by-layer and then WL-by-WL within each layer, a cell may be disturbed up to two times. For a random programming order, a cell may be disturbed up to four times. The disturbance effects accumulate and lead to errors if $\Delta V_{(t)}$ becomes too larger.

Given that capacitance between cells depends on their distance, this type of program disturbance is projected to increase dramatically with technology scaling. The disturbance was obviously observed for planar flash at $1\times$-nm technology node and below, which has become one of the major scaling challenges [19]. For 3-D flash chips that currently adopt larger technology node (> 40 nm), the disturbance begins to manifest as being significant [6], [7]. While cell optimizations help to mitigate its severity temporarily [11], [20], with 3-D integration fast approaching its stacking limits, future 3-D flash chips have to scale to smaller technology node, which face severe disturbance, similar as that in 2-D flash.

2) Other Disturbance Sources: There exist two more types of disturbance. One is the channel coupling-based program disturbance. For example, when programming cells A and D in Fig. 1, other cells on the same WL, e.g., cell E, referred to as inhibited cells, may be disturbed. $V_{\mathtt{prog}}$ and V_{cc} are applied to the WL and BL of each inhibited cell, respectively. The inhibited channel is boosted during programming to avoid unintentional threshold voltage shift. Unfortunately, one WL is physically spliced together across multiple layers along Z direction, thus 3-D stacking leads to more disturbance errors on inhibited cells [2].

The other disturbance comes from flash read operations. For the unselected cells in the same block during read, their WLs and BLs are applied $V_{\mathtt{pass}}$ and GND. While one read contributes little disturbance, a 3-D flash block contains much more pages than its 2-D counterpart such that there is large accumulated disturbance for long lived hot pages.

The significance of these two types of disturbance may amplify with technology scaling. We leave the design of effective schemes on mitigating them in our future work.

B. Approximate Computing

Approximate computing is an emerging computing paradigm that exploits the inherent error resilience in many modern applications, such as audio, video, and image processing applications [21]. For example, it is often not noticeable to have a small number of errors in rendering video streams; and having a small number of bad pixels in high resolution images usually does not affect image classification. In order to ensure the overall quality of service for these applications, data is often divided into *critical data* and *noncritical data*, with the help from either the programmer [21] or a compiler [22]. While errors may appear in noncritical data, the critical data are kept precise.

Most of approximate computing proposals exploit error-resilience in cache and main memories and/or within approximation-aware microarchitecture. Liu *et al.* [21] proposed to reduce the refresh frequency of DRAM banks that save noncritical data, which not only saves refresh energy but also improves performance. Esmaeilzadeh *et al.* [23] developed approximation-aware hardware to efficiently support approximate programming model.

Extending approximate computing to solid state memories enables the construction of approximate storage. Sampson *et al.* [9] proposed to use fewer steps when programming MLC PCM, which can be extended to MLC flash to improve the performance of writing noncritical data. Guo *et al.* [24] observed that the bits of encoded image have nonuniform error resilience and proposed to store them in separate regions with different precision guarantee. Jevdjic *et al.* [25] computed bit-level reliability requirements for encoded videos, and showed that reliability can be traded for density improvement in video storage.

A main difference between approximation storage schemes and ApproxFTL is that the former achieves performance and energy saving benefits by speeding up the processing of approximation data. ApproxFTL further targets at reducing program disturbance in physical blocks so that it benefits the processing of both precise and approximate data.

A critical problem in approximate computing is to evaluate the quality of the approximate output. Early designs adopted static evaluation and sampling schemes. The recent advances proposed online evaluation. For example, Khudia *et al.* [26] proposed to dynamically monitor output quality and adjust computation accuracy accordingly. This is orthogonal to our design. Currently, ApproxFTL adopts pessimistic static evaluation, which can be improved by online error monitoring for better performance.

C. Flash Translation Layer

The FTL is a critical component in NAND flash-based SSDs. It has been extensively studied for improved system performance and extended lifetime of SSDs. Most FTL designs [27], [28] partition hot and cold data, which helps to minimize write amplification. By integrating data compression [29] and de-duplication [30], the SSD lifetime may be prolonged with reduced flash writes. Optimizing GC policies in FTL [31] helps to improve the average response time for flash systems.

ApproxFTL differs from these FTL designs in that it leverages approximate write to mitigate program disturbance in 3-D flash and improve the performance.

III. Approximate Write on 3-D Flash

Approximate computing is an emerging computing paradigm that requires the coordination across software and

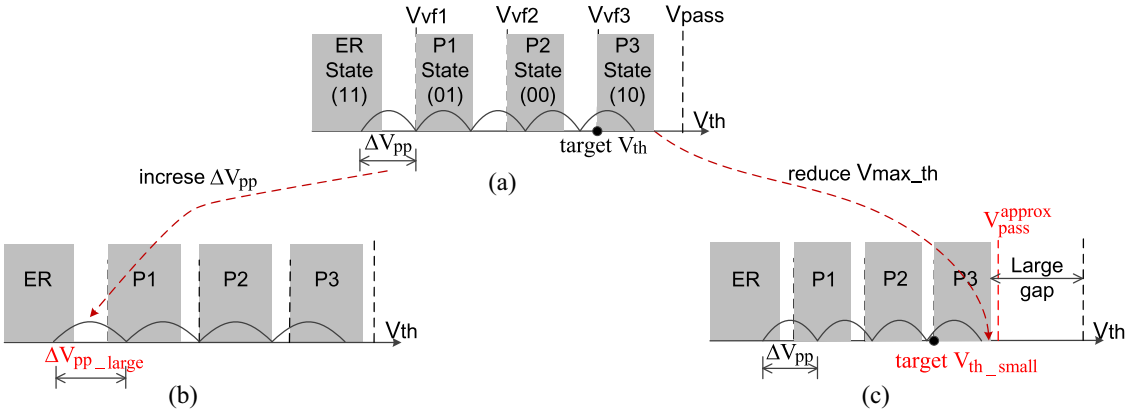


Fig. 2. ISPP schemes for (a) baseline (i.e., precise page programming); (b) approximate programming with large ΔV_{pp}; and (c) approximate programming with reduced threshold voltage $V_{\mathtt{max\_th}}$.

hardware layers. In particular, previous approximate computing approaches differentiate precise data and approximate data, which may be annotated either by the programmers or using an approximate computing aware compiler, e.g., Enerj [9]. ApproxFTL makes the similar assumption in implementing the flash memory-based approximate storage—the annotation is passed together with each write request to the host interface logic of the SSD.

A. ISPP-Based Flash Programming

Flash programming widely adopts the incremental step pulse programming (ISPP) strategy. A typical ISPP scheme consists of multiple steps with each step applying an increasingly biased programming voltage to raise the voltage of the cells being programmed. The programming voltage increment at each step, referred to as ΔV_{pp}, is often a constant. After each step, a verify operation is carried out to check if the cell voltage is above the target voltage level (target V_{th}) and determine if the programming process can be terminated. Fig. 2(a) illustrates the four target reference voltages of 2-bit MLC cells. The shaded range indicates the voltage window for each state while the white gap between two states indicates the guard band.

Studies have revealed that flash reliability and access performance depend on the adopted ISPP scheme. Given that cell voltage may fluctuate after programming due to disturbance and charge leakage, the voltages of a small number of flash cells may deviate into the ranges of their neighboring states, resulting in raw bit errors (RBEs) that demand ECC code to fix before returning the data to the user. On one hand, the more bit errors a flash page contains, the longer error correction process it takes during read, and the lower the read performance is. On the other hand, choosing a smaller ΔV_{pp} helps to achieve narrower voltage range for each state and wider guard bands, i.e., the data are more reliable. Due to larger guard bands, the same voltage shift leads to fewer errors so that the read is faster. However, it takes more steps and longer latency to finish the write operation.

B. Approximate Write for 3-D Flash

Based on the interdependency between flash reliability and access performance, a simple implementation of approximate write is to use larger ΔV_{pp} than that in the baseline, which reduces the number of write steps and improves the write performance [9].[2]

As shown in Fig. 2(b), adopting a larger ΔV_{pp} shrinks the guard bands between MLC states, which leads to a significant increase of RBE rates (RBERs). Our experiments show that the RBER changes from 10^{-8} when using conventional precise programming to 7.2×10^{-4} when using a large ΔV_{pp}, which is based on the split-page 3-D VG NAND Flash chip [14] to carry out these Monte Carlo simulations (see Section III-C for more details). Given that such high RBER is beyond the correction capacity of an ECC code with reasonable space overhead, the ECC correction phase for a read operation may be skipped [32]. The quality of the output is ensured either by static analysis [22] or online error monitoring [26]. In summary, approximate write improves both read and write performance by trading the data accuracy.

In the following discussion, we use *precise/approximate write* to denote the ISPP that does-not/does relax data reliability, respectively. That is, we expect $10^{-8}/7.2 \times 10^{-4}$ RBER, respectively. We use *precise/approximate page* to denote the data left in the physical page after precise/approximate write, respectively.

The simple approximate write, while improving the access performance to approximate data, has limited impact on accessing precise data. More importantly, it lacks the ability to mitigate the challenging program disturbance problem in 3-D flash.

In this paper, we adopt an alternative approximate programming, which reduces the maximal threshold voltage $V_{\mathtt{max\_th}}$, and accordingly the reduced target voltage level $V_{\mathtt{th\_small}}$ and the guard bands, keeping the same ΔV_{pp} as in the baseline, as shown in Fig. 2(c). Comparing to the scheme using

[2]Sampson *et al.* [9] proposed to reduce the number of write steps for PCM, here we adapt it to 3-D flash for comparison.

large ΔV_{pp}, our $V_{\texttt{max\_th}}$-reduction-based approach has the following advantages.

1) Reducing $V_{\texttt{max\_th}}$ helps to reduce program disturbance in physical blocks. According to (1), the program disturbance depends on the cell voltage change before and after programming. Reducing $V_{\texttt{max\_th}}$ reduces the gratitude of the voltage change, indicating mitigated disturbance on the victim cells. This paper shows that we may reduce $V_{\texttt{max\_th}}$ by up to 38% while maintaining comparable RBER on approximate data as that using larger ΔV_{pp}, i.e., the RBER increases from 10^{-8} when using conventional precise programming to 7.2×10^{-4} in our approximate programming (see Section III-C for more details). Programming an approximate page with reduced $V_{\texttt{max\_th}}$ reduces its disturbance to neighboring cells proportionally, i.e., by up to 38% reduction.
2) Reducing $V_{\texttt{max\_th}}$ helps to improve the write performance when writing either precise or approximate pages. Approximate write becomes faster because we reduce the guard bands so that it takes fewer steps to finish the ISPP programming. Whether we can speed up a precise write operation depends on how we are to program its neighboring pages. Let us assume pages are sequentially programmed in the physical block, and we use (*, j, k) to denote a flash page in layer k with page index j.
   a) In the case if we may potentially write precise pages to (*, j+1, k) and (*, j, k+1), we have to be conservative in programming page (*, j, k), that is, it has the same performance as that in the baseline.
   b) In another case, we may have decided only to allocate approximate pages to (*, j+1, k) and (*, j, k+1). Given that programming (*, j+1, k) and (*, j, k+1) exhibits 38% less disturbance on (*, j, k), we may slightly relax the reliability in writing precise page (*, j, k). We observe around 24% performance improvement.
   c) In yet another case, we may decide to write those neighboring pages first and write (*, j, k) the last. For page (*, j, k), there is no future disturbing operation, while in traditional sequentially program case both (*, j+1, k) and (*, j, k+1) exhibit disturbance on page (*, j, k). We may more aggressively relax the reliability in writing precise page (*, j, k) than that of case b). We may achieve maximized write relaxation, which is about 33% faster than the baseline. Given that the performance improvement of precise write depends on how the pages are allocated in the physical block, we are motivated to develop approximate-write aware data allocation. We will elaborate it in details in the next section.
3) Reducing $V_{\texttt{max\_th}}$ helps to achieve less stress for the cells from an approximate page. Studies have shown that the wearing effect of flash memory depends on the erase voltage at erase time [33], [34]. When there is a reduction of $V_{\texttt{max\_th}}$, we can reduce erase voltage proportionally such that the P/E cycle can be improved.

Reducing the maximal threshold voltage method has also attracted research attention by other previous works, which are used for traditional precise storage with improved performance/lifetime [34], [35]. But they need to lower the $V_{\texttt{max\_th}}$ voltage conservatively, where data are accommodated by the flash error correction capability. We would like to reduce the $V_{\texttt{max\_th}}$ voltage more aggressively in approximate write scenario, where its RBER may exceed the error correction capability. Moreover, using approximate-write operations to store error-resilient data of modern applications, we propose ApproxFTL, an approximate-write aware FTL design to explore above advantages with different page allocation patterns. We will elaborate it in details in Section IV.

A drawback of reducing $V_{\texttt{max\_th}}$ is that it reduces the retention reliability. In our design, only approximate pages may be affected, i.e., suffer from an increment of retention errors. Given approximate pages save noncritical data, our experimental study shows its impact is negligible. For precise pages, we use the same $V_{\texttt{max\_th}}$ as the baseline to avoid the retention issue for critical data.

It should be noted that approximate computing blurs the boundary for excluding traditionally defective writes. The RBERs of writing precise and approximate computing are 10^{-8} and 7.2×10^{-4}, respectively, exhibiting more than $1000\times$ difference. The RBER of the approximately written cells, when considering the process variations and suffering from the worst-case write disturbance and retention charge loss would go beyond LDPC correction capability, which fails application execution under the traditional computing paradigm. However, the errors can generate acceptable application outputs under the approximate computing paradigm.

C. Modeling Details

In this section, we present the details of reliability model under different maximal threshold voltages. The following formulas are extended from [36], but different from [36], the 3-D flash reliability model exploits program disturbance coming from Z direction.

At first, threshold voltage distribution of erase $P_e(x)$ and programmed states $P_p^{(k)}(x)$ can be expressed as

$$P_e(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

$$P_p^{(k)}(x) = \begin{cases} \frac{1}{\Delta V_{pp}}, & \text{if } V_p^{(k)} \le x \le V_p^{(k)} + \Delta V_{pp} \\ 0, & \text{other.} \end{cases} \tag{3}$$

Program disturbance increases the threshold voltage of the victim cell systematically, where its threshold voltage shift is modeled in (1). Its threshold voltage distribution follows:

$$P_{pd}(x) = \begin{cases} \frac{\alpha}{\sigma_d\sqrt{2\pi}} e^{-\frac{(x-\mu_d)^2}{2\sigma_d^2}}, & \text{if } |x-\mu_d| \le w \\ 0, & \text{other} \end{cases} \tag{4}$$

where the BL-to-BL and diagonal coupling ratio are often small and neglected, as explained before, and WL-to-WL and layer-to-layer coupling ratio is γ_y and γ_z, respectively.

The probability density function of threshold voltage with P/E-induced fluctuation, and retention time-induced fluctuation can be expressed as

$$P_{pe}(x) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}} \tag{5}$$

$$P_{rt}(x) = N\left(\mu_r, \sigma_r^2\right). \tag{6}$$

Then the final threshold voltage distribution of the kth programmed state is obtained as

$$P^{(k)}(x) = P_p^{(k)}(x) \otimes P_{pd}(x) \otimes P_{pe}(x) \otimes P_{rt}(x). \tag{7}$$

Therefore, RBER model can be computed as

$$\text{RBER} = \sum_{k=0}^{q} \left(\int_{-\infty}^{V_p^{(k)}} p^{(k)}(x)dx + \int_{V_p^{(k+1)}}^{+\infty} p^{(k)}(x)dx \right). \tag{8}$$

UBER is typically required to be under 10^{-15} [39], and the relationship between UBER and RBER is

$$\text{UBER} = \frac{\sum_{n=t+1}^{N_{cw}} \binom{n}{k} \cdot \text{RBER}^n \cdot (1 - \text{RBER})^{(N_{cw}-n)}}{N_{\text{User}}}. \tag{9}$$

The variables and notations in the reliably model are leveraged from [36] and [37]. According to the JEDEC standard JESD47I.01 [38], floating gate/charge trap flash memory needs 1000 h retention time in high-temperature retention bake, and the 10K maximal P/E cycling for MLC flash memory. On the basis of 3-D VG NAND flash memory [14], the program disturbance contributes 0.2 and 0.15 shift through Y and Z direction, respectively, whereas we set ΔV_{pp} and $V_{\texttt{max\_th}}$ as 0.2, and 8, in the conventional ISPP scheme [Fig. 2(a)]. And in case of programming with reduced threshold voltage $V_{\texttt{max\_th}}$ [Fig. 2(c)], we aggressively set ΔV_{pp} and $V_{\texttt{max\_th}}$ as 0.2, and 5, respectively.

IV. ApproxFTL Design

In this section, we present an overview of ApproxFTL and elaborate the three enhancements that we integrate at the FTL level.

A. Overview

ApproxFTL is an approximate-write aware FTL that exploits $V_{\texttt{max\_th}}$-reduction-based approximate page programming for maximized disturbance reduction and performance improvement in 3-D-flash-based SSDs.

An overview of ApproxFTL is shown in Fig. 3. At the high level, an error-resilient application has its data partitioned into critical region and noncritical region, with the assistance from either manual tagging or compilation analysis [22]. The file system is enhanced to pass the approximate tag to the FTL in the SSD. That is, an I/O request sent to flash FTL is a quadruple

`(LPN, Size, Read/Write, ApproxFlag)`

where `LPN` is the starting logical page number of the data block, `Size` is the length of the data block, `ApproxFlag` denotes if the user data can be written using approximate write. FTL then strips the data block to multiple chips for achieving chip-level parallelism.

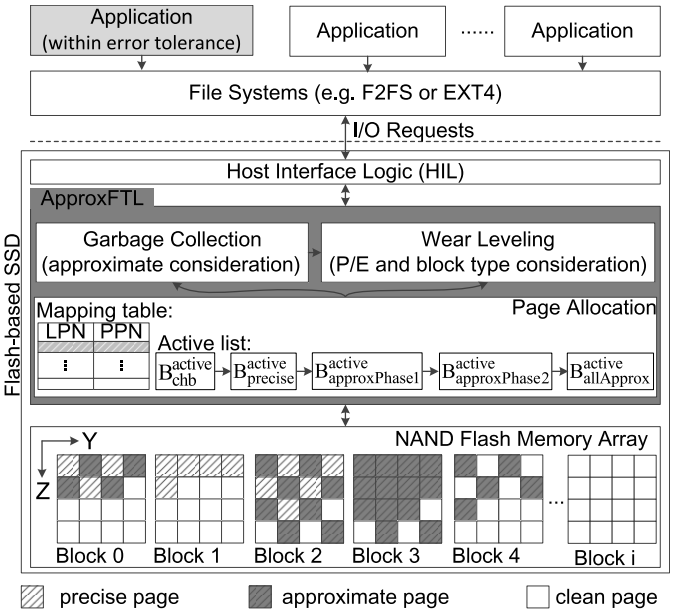


Fig. 3. Overview of ApproxFTL.

At the low level, an approximate-write controller is integrated inside the flash module. It receives the requests together with approximate tag from the enhanced FTL, and conducts the corresponding approximate and precise writes with enhanced ISPP programming.

At the middle level, we enhance the baseline FTL to exploit the disturbance mitigation inside each physical block. We first enhance the baseline page allocator, the component in FTL for assigning physical page number (PPN) to each incoming page write with its LPN. We develop a pattern-guided page allocator that determines the allocation pattern before allocation. By taking advantage of the pattern information, ApproxFTL effectively minimizes disturbance and speeds up both approximate and precise page writes.

We then enhance the baseline wear leveling algorithm with approximate-write awareness. Since page wearing effect depends on its maximal threshold voltage, we differentiate approximate and precise, track wearing effect based on block usage, and strive to proportionally distribute both types of write requests to each page.

We also enhance the baseline GC to mitigate program disturbance. At the GC time, ApproxFTL moves valid approximate and precise pages in two batches to active blocks for maximized disturbance reduction. In addition, we promote approximate pages to precise ones to prevent introducing unbounded precision errors.

Next, we elaborate the three major enhancements in ApproxFTL.

B. Pattern-Guided Page Allocation

In Section III, we have presented that the programming of precise pages can be speeded up. This is enabled when their neighboring locations are programmed with approximate pages. This motivates our pattern guided allocation strategy—when fetching a clean block for LPN-to-PPN mapping, we

predetermine its page allocation pattern, i.e., how approximate and precise pages are laid out in the block, and then strictly follow the pattern in the future mapping.

1) Page Allocation Pattern: In this paper, we adopt the following three page allocation patterns while more patterns will be exploited for further optimization.

a) Checkerboard pattern: When there are about the same number of precise and approximate page writes, we allocate them interleavingly in both *Y* and *Z* directions, similar to a checkerboard, e.g., block 0 in Fig. 3. We next discuss the disturbance reduction in the block by considering two different write orders.

The first write order is sequential interleaving—starting from the first page, we take approximate and precise writes *alternatively* and map them to physical locations in sequential order. By adopting this order, a precise page, e.g., page *s* that was just programmed, may be disturbed at most by two future approximate writes, one in *Y* direction and one in *Z* direction. In this case, the precise write can be speeded up, as discussed in Section III. For an approximate page, it may be disturbed by up to two precise writes, which is the same as the baseline. The disturbance on approximate pages has little influence.

The second write order is two-phase interleaving—we first write all approximate pages and then write the precise pages to the block. During the phase of writing approximate pages, we skip locations that are allocated to precise pages. As a result, by the time when we write a precise page, all its neighboring pages (approximate pages) have been programmed. Since there is no future program disturbance to the precise page, we can speed up precise write by a larger amount comparing to that using sequential interleaving write order.

b) All precise pattern: If there are significantly more precise page writes than approximate ones, or precise page writes arrive in a burst, we need to fetch physical blocks for precise writes only. We program pages in the block in sequential order such that each programmed page may be disturbed by two precise writes (at later times). This is the same as the baseline.

For this pattern, we take the sequential order by pessimistically assuming all previously programmed pages are alive when we program the last page of the block. Other writing orders [6] may achieve better disturbance mitigation and performance improvement.

c) All approximate pattern: If there are significantly more approximate page writes, we allocate physical blocks and map only approximate writes to them. Similar as above, we choose the sequential order when writing pages to the block.

2) LPN-to-PPN Mapping: We next elaborate the enhancement for the mapping from an incoming write (with LPN) to a PPN. Setting up the mapping is often simple in existing FTLs: the FTL keeps one active physical block, maps incoming writes (with different LPNs) to consecutive PPNs in the block, and fetches a new block after depleting the current one. With the recent advance in flash industry, e.g., the multistream technology in Samsung 840 Pro SSD [39], [40], the FTL maintains more than one active blocks such that a write can be mapped to a better choice.

Since our pattern-guided page allocation restricts the capability of mapping an incoming write to an active block, we propose to maintain at most five active blocks with different allocation patterns such that an incoming write can always be directed to the best block. The small number of active blocks has little impact on performance as shown in [39].

As shown in Fig. 4, we keep the following active blocks. A block is always tagged as being clean after erase, i.e., $B^{\texttt{clean}}$.

1) The $B^{\texttt{active}}_{\texttt{chb}}$ block. This block is to take the checkerboard page allocation pattern, and the sequential interleaving write order. That is, it services approximate and precise writes alternatively. For example, if the last LPN page mapped to the block is an approximate write, the FTL has to map a precise LPN page before mapping another approximate page. After programming the last page of $B^{\texttt{active}}_{\texttt{chb}}$, the FTL converts it to $B^{\texttt{used}}_{\texttt{chb}}$ and allocates a clean block to $B^{\texttt{active}}_{\texttt{chb}}$.
2) The $B^{\texttt{active}}_{\texttt{precise}}$ block. This block is to take the all precise pattern and sequential write order. This is the same as the baseline approach. The $B^{\texttt{active}}_{\texttt{precise}}$ block is converted to $B^{\texttt{used}}_{\texttt{precise}}$ after its last write, and then the FTL allocates a clean block to $B^{\texttt{active}}_{\texttt{precise}}$.
3) The $B^{\texttt{active}}_{\texttt{approxPhase1}}$ block. This block is to take the checkerboard pattern and two-phase interleaving write order. The block is currently in the first phase, i.e., servicing approximate writes. Precise writes cannot be sent to this block.
4) The $B^{\texttt{active}}_{\texttt{approxPhase2}}$ and/or $B^{\texttt{active}}_{\texttt{allApprox}}$ block. When $B^{\texttt{active}}_{\texttt{approxPhase1}}$ finishes the first phase, the FTL converts it to either $B^{\texttt{active}}_{\texttt{approxPhase2}}$ or $B^{\texttt{active}}_{\texttt{allApprox}}$. The former indicates that the block is now servicing precise writes while the latter indicates that the block continues to service approximate writes. After the above conversion, the FTL moves a clean block to $B^{\texttt{active}}_{\texttt{approxPhase1}}$. Similar as above, the $B^{\texttt{active}}_{\texttt{allApprox}}$ block is tagged as `used` after its last write. Note that $B^{\texttt{active}}_{\texttt{approxPhase2}}$ will be tagged as $B^{\texttt{used}}_{\texttt{chb}}$ after its last write, because it takes the checkerboard pattern as well.

At any given time, we have up to five active blocks—we always have one of the first three types and may be missing one or both of the last two active blocks types. Fig. 4 summarizes the life cycle of a physical block in ApproxFTL.

3) Choosing the Best Block: Given an approximate or precise write may be serviced by more than one blocks, Algorithm 1 elaborates the details on how to choose the best one. For the writes from the file systems, we first check if it is the precision type that $B^{\texttt{active}}_{\texttt{chb}}$ wants and gets serviced if it is a match. Otherwise, we direct approximate writes to $B^{\texttt{active}}_{\texttt{approxPhase1}}$ and $B^{\texttt{active}}_{\texttt{allApprox}}$, and precise writes to $B^{\texttt{active}}_{\texttt{approxPhase2}}$ and $B^{\texttt{active}}_{\texttt{precise}}$.

When we have all five active blocks, we pause mapping approximate writes to $B^{\texttt{active}}_{\texttt{approxPhase1}}$ until one of $B^{\texttt{active}}_{\texttt{approxPhase2}}$ or $B^{\texttt{active}}_{\texttt{allApprox}}$ depletes.

When we miss $B^{\texttt{active}}_{\texttt{approxPhase2}}$ and have all other active blocks, we prioritize the mapping of approximate pages to $B^{\texttt{active}}_{\texttt{approxPhase1}}$ (rather than $B^{\texttt{active}}_{\texttt{allApprox}}$). The goal is

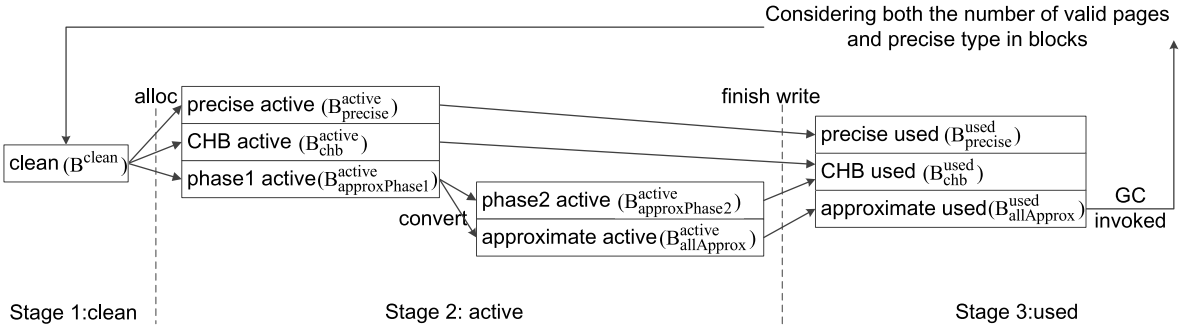


Fig. 4. Life cycle of flash memory blocks.

Algorithm 1: Active Block Selection

Input: Write requests.
Output: $B_{current}$, the selected active block.

repeat
  if *request.ApproxFlag*==B^{active}_{chb}.*want* **then**
    $B_{current}$=B^{active}_{chb} ;
  else if *request.ApproxFlag*==*precise* **then**
    if $B^{active}_{approxPhase2}$ *exist* **then**
      $B_{current}$=$B^{active}_{approxPhase2}$;
    else
      $B_{current}$=$B^{active}_{precise}$;
    end
  else
    if $B^{active}_{approxPhase2}$ *no exist* **then**
      $B_{current}$=$B^{active}_{approxPhase1}$;
    else
      $B_{current}$=$B^{active}_{allApprox}$;
    end
  end
until *no write request*;

to proceed to the end of the first phase early such that $B^{active}_{approxPhase1}$ can be converted to $B^{active}_{approxPhase2}$ to service precise writes.

When we only have the first three block types. $B^{active}_{approxPhase1}$ is by default converted to $B^{active}_{approxPhase2}$ when it reaches the end of the first phase.

4) Separating Hot and Cold Data: The benefits of separating hot and cold data have been evaluated in several previous studies [41]–[50]. In this paper, this separation strategy is also implemented to improve the GC performance, and we categorize the hot/cold requests according to their read or write data sizes, i.e., small-sized approximate/precise requests are treated as hot ones. The size-based strategy was widely adopted to classify hot and cold data in recent studies [41]–[44]. Many such small-sized requests process metadata and thus are critical to system performance. While recent designs confirmed the effectiveness of this metric, data hotness can be defined differently [45]–[50]. The proposed designs in this paper are independent of the hotness definition and can be adapted to use other hotness metrics. All of the flash blocks are separated into two allocation pools, one hot pool for hot data and another cold pool for cold data. The hot data will be navigated to blocks in the hot pool, and vice versa.

C. Wearing Leveling

In recent studies, Jeong *et al.* [33] and Shi *et al.* [34] revealed that reducing the maximal threshold voltage V_{max_th} enables the adoption of a lower erase voltage, which reduces the *effective wearing*. For example, in [34], when reducing V_{max_th} from 4.25 to 3.40 V, erasing the block 100 times with lower voltage corresponds to 80 erases with the default voltage, i.e., the *effective wearing* is 0.8.

In this paper, we extend *effective wearing* to take advantage of the wearing benefits from approximate writes. In particular, for the blocks that only save approximate data, all pages share the same reduced V_{max_th} so that a lower erase voltage can be employed, which exhibits proportional *effective wearing* reduction, i.e., 0.62 of the baseline. For other blocks, we use the default erase voltage as at least some pages in these blocks have no V_{max_th} reduction.

ApproxFTL tracks the *effective wearing* of all physical blocks. Given a block b_i, ApproxFTL updates its *effective wearing* W_{b_i} at erase time using (2). We only exploit the blocks that save approximate data in this paper

$$W_{b_i} += \begin{cases} 0.62 & : \text{ if } b_i \text{ is a } B^{used}_{allApprox} \text{ block} \\ 1.00 & : \text{ otherwise} \end{cases} \quad (10)$$

while W helps to track the effective wearing at the block level, cells within a physical block have different wearing effects if the block adopt checkerboard page allocation pattern. Instead of developing expensive schemes to track at cell level, ApproxFTL takes a simple strategy that flips the type of page that can be allocated to its first physical page. That is, the first physical page, if it is mapped to service an approximate write this time, would service a precise write next time. The flag is checked, used and flipped only when the block is assigned to use checkerboard pattern. Using the block with other patterns leads to uniform stress of all cells.

At runtime, ApproxFTL tracks the effective wearing W of all physical blocks based on their programming pattern, and ideally returns the clean block with the smallest W if a new clean block is requested. We found it is often not necessary to track the smallest W, in the experiments, we identify the smallest W periodically and, when a clean block is requested,

conduct a greedy search and return the first block whose W is not two times bigger. Our results show that this simple heuristic performs equally well while greatly reducing the sorting overhead.

D. Garbage Collection and Refresh

ApproxFTL converts active blocks to three types of *used* blocks after programming their last pages (Fig. 4). When there is a need to invoke GC, a traditional collector would reclaim the block with the smallest number of valid pages, regardless of their page types, i.e., if these pages are approximate or precise pages. Such a garbage collector often leads to suboptimal results.

We enhance the baseline GC as follows.

1) *Victim Block Selection Policy:* At the time to identify a victim block, we not only consider the total number of valid pages but also their approximate type. For blocks with the same number of valid pages, we prioritize the selection of blocks that have fewer valid precise pages. This is because, when moving the same number of valid pages, the smaller the number of precise pages is, the smaller the overall disturbance the move introduces to the system. In particular, if there is no active block $B^{active}_{approxPhase2}$, we demote the selection of $B^{used}_{precise}$ blocks as the latter contains all precise blocks. Moving valid pages from the block would copy them to $B^{active}_{precise}$, which has large overall disturbance.
2) *Valid Page Copying Policy:* When copying valid pages from the victim block, we first move all valid approximate pages and then the precise pages. Copying approximate pages in batch helps to create $B^{active}_{approxPhase2}$ if it does not exist. Valid precise pages can then be copied to this block rather than $B^{active}_{precise}$, which introduces minimized program disturbance.
3) *Approximate Page Upgrade Policy:* Since writing approximate pages tends to introduce uncorrectable errors in the page, copying these pages to new locations, if keeping using approximate write, may accumulate significant large amount of errors that fail the application. For this reason, we attach a flag to each block to record the largest number of approximate writes to any given page. We promote all approximate pages of the victim block to precise page if this flag reaches the threshold. For example, we set the flag of a block B_1 to 1 if the approximate pages of this block are written with data from the file system. Garbage collecting this block copies its approximate pages to a new block B_2, we set the B_2's flag to 2 if it is not bigger. Assuming our promotion threshold is 2, garbage collecting B_2 at a later time shall promote all approximate pages to precise ones, which helps to prevent introducing unbounded errors to the approximate pages.

An approximate page accumulates retention errors after being programmed. To prevent an approximate page from suffering too many errors that fail the application, we refresh each approximate page in six months. ApproxFTL provides the worst-case reliability guarantees under both programming errors and retention errors. Given that we cannot distinguish the error types, retention errors cannot be corrected at refresh time. We therefore upgrade approximate pages to precise pages when refreshing such pages.

E. Overhead Analysis

AproxFTL, while mitigating program disturbance and prolonging chip lifetime, introduces three types of overheads: 1) hardware overhead; 2) storage overhead; and 3) firmware overhead. For the hardware overhead, we need the hardware support to enable approximate write. Similar approaches have been adopted in [9] with reasonable overhead.

For the storage overhead, we need three flags as follows.

1) A 2-bit flag to indicate the page allocation pattern.
2) A 1-bit flag to indicate the type of the first page when the last time the block was used as $B^{used}_{approxPhase2}$ or B^{active}_{chb}. This flag is to assist intrablock wear leveling. Note, these three bits help to identify the block type of the to-be-programmed block. We choose reduced erase voltage to erase a block if all its pages are programmed as approximate pages, and choose normal erase voltage otherwise.
3) A 3-bit flag to indicate the maximal time that an approximate page in the block has been written using approximate write. This flag is to assist approximate page promotion. The seven bits are stored at the out-of-band region of the first page for each block, which has negligible overhead for modern flash chips.

For the firmware overhead, we demand five active blocks, i.e., the integration of multistreamed technology if it is not embedded already. Kang *et al.* [39] showed that the overhead is negligible.

V. Performance Evaluation

In this section, we present the experimental methodology, evaluate the effectiveness of ApproxFTL, and analyze the experimental results with comparison to the state-of-the-art.

A. Experiment Setup

To evaluate the effectiveness of our proposed ApproxFTL, we implemented it in an event-driven SSD simulator. We simulated a 128 GB eight-layer 3-D NAND flash-based SSD [14], where the pages from one block are spread across all eight layers, each layer has 64 pages from the block, and each page is of 8 KB. The GC is triggered when the number of free blocks goes below 10% of the total number of blocks. The GC operations are executed in the background in order to minimize the influence on the foreground requests. The key NAND flash parameters used in our simulation are listed in Table I.

In the baseline, the FTL adopts sequential write order and program all pages using precise writes. A programmed page, even after two disturb operations from its neighboring pages in Y and Z directions, respectively, has 10^{-8} or better RBER. The latency for read, program, and erase operation is 45 us, 700 us, and 3.5 ms, respectively.

TABLE I
KEY PARAMETERS OF NAND FLASH MEMORY

Flash Organization
density = 128 GB, total number of layers = 8, page size = 8 KB, page per block = 512, total blocks = 32768, GC threshold = 10%, over-provisioning ratio = 7%, page read = 45 *us*, page write = 700 *us*, block erase = 3.5 *ms*
Electrical Feature
DRAM R/W current = 125 mA, Flash R/W/E current = 25 mA, supply voltage = 3.3 V

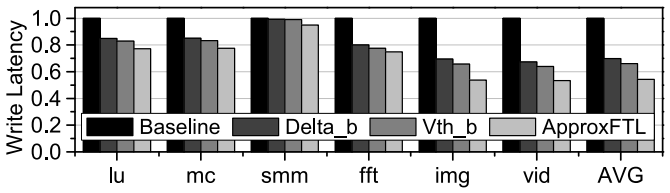


Fig. 5. Normalized write performance.

1) Workloads: We used two types of workloads in the evaluation. First, the raw error-tolerance I/O traces is collected. In addition, we plugged in errors to random locations of the approximate data and evaluated the quality degradation of the output.

When running four scientific and numerical computing applications on a Centos-7 Linux box—*fft*, *smm*, *lu*, and *sor*, we adapt the annotated type system, EnerJ [9], to distinguish between approximate and precise data types, and at the same time we use the *blktrace* tool to capture the block level I/O traces. With type annotations, we can calculate the proportion of approximate noncritical data. Then approximate data types can be distinguished from the collected I/O traces. When application names are not these scientific computing applications in the I/O traces, e.g., swapper, its requests are regarded as traditional precise data. Data requests with the same application names are isolated the precise portion from the approximate portion in the equal proportion for simplicity. We also collected the I/O traces for one video and one image processing application treated as the same way—*img* and *vid*.

2) Output Quality Metrics: To evaluate the quality of the output, we adopted application-specific metrics, the same as these in previous works [9]. Since previous six I/O traces do not have the detailed data content for each request, public data sets are used for output quality evaluation. In particular, *dt* uses a decision tree to predict the context in the data set "sensorlog." *Svm*, which is a supervised learning model based on a support vector machine, and *ann* which is another trained classifier based on a feed-forward neural network, analyze the "pendigits" data set. And *knn*, *k*-nearest neighbor algorithm is used to classify multiclass subgenus of "Iris" data set. The quality metric in these classifiers is the classification accuracy on test datasets relative to the real classification results, which is consistent with previous works [9].

3) Performance Metrics: We evaluated ApproxFTL using five metrics, average read response time, average write response time, wearing, total energy consumption, write amplification, and quality metric. The average read/write response time is a good metric for estimating ApproxFTL performance. And, the efficacy of the proposed scheme is also evaluated-based the wearing for each FTL, which specifies the lifetime improvement under different FTLs, i.e., the wearing is more severe, then the lifetime of flash memory will be shorter. The total energy consumption measure is used to evaluate the impact of FTLs on energy consumption. All operations in flash-based SSD consume energy. For example, given that the time consumption of one flash erase operation is 3.5 ms, whereas the supplying voltage is 3.3 V and the erase current is 25 mA as listed in Table I, one erase operation in flash consumes 288.75 μJ. The write amplification metric is used to evaluate the impact of GC on internal writes. The quality metric is evaluated based on quality degradation to demonstrate the effect of approximate storage on application-specific quality loss.

4) Schemes for Comparison: In the experiments, we implemented and compared the following schemes.

1) `Baseline`: This is the scheme that implements the traditional page-mapping FTL [51]. The baseline adopts sequential write order when programming data in the active block, and programs all the pages using traditional precise writes, i.e., t_PROG=700, t_R=45, t_BERS=3500 [52]. It does not exploit the error resilience in modern applications.
2) `Delta_b`: This is the scheme that implements the simple approximate write [9]. It uses a larger $\Delta V_{\mathtt{pp}}$ that is 1.5× of the baseline, and assumes page writes from upper level are tagged so the FTL knows if it needs approximate write or precise write. The RBER jumps to 7.2×10^{-4} from the baseline.
3) `Vth_b`: This is the scheme that implements our $V_{\mathtt{max\_th}}$-reduced approximate write. The $V_{\mathtt{max\_th}}$ is around 62% of the baseline. The RBER jumps to 7.2×10^{-4} from the baseline.
4) `ApproxFTL`: This scheme is built on top of `Vth_b`. It integrates three enhancements in FTL to maximize disturbance mitigation.

B. Write Response Time Comparison

Fig. 5 compares the write response time from different schemes. The results were normalized to `Baseline`. From the figure, `ApproxFTL` reduces the average write response time by 45.64% over `Baseline`. The write performance improvement comes from: 1) due to the reduction of ISPP steps in `ApproxFTL`, `Delta_b` and `Vth_b`, the response time of approximate write is reduced and 2) due to reduced disturbance, programming a precise page takes less time if it is mapped to $\mathrm{B}^{\mathtt{active}}_{\mathtt{approxPhase2}}$ and $\mathrm{B}^{\mathtt{active}}_{\mathtt{chb}}$. In summary, `ApproxFTL` achieves the largest reduction, i.e., 46.67%, in *vid*, and the smallest reduction, i.e., 5.15%, in *smm*.

To fully understand the write performance improvement in `ApproxFTL`, Fig. 6 reports the cumulative distribution of different active block instances. We ignore $\mathrm{B}^{\mathtt{active}}_{\mathtt{approxPhase1}}$ as it is an intermediate state, which gets converted to either $\mathrm{B}^{\mathtt{active}}_{\mathtt{approxPhase2}}$ or $\mathrm{B}^{\mathtt{active}}_{\mathtt{allApprox}}$. Since writing pages in $\mathrm{B}^{\mathtt{active}}_{\mathtt{precise}}$ is the same as the baseline, the more the pages are

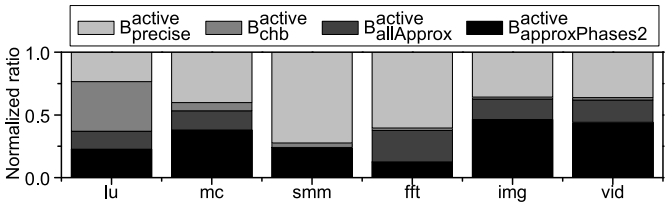


Fig. 6. Cumulative distribution of block instances in ApproxFTL.

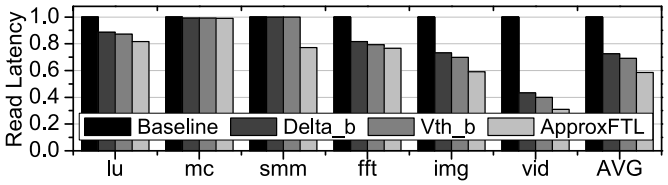


Fig. 7. Normalized read performance.

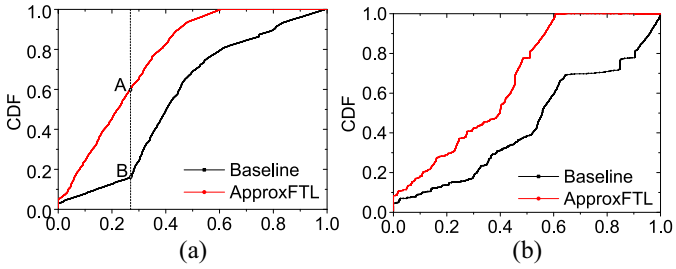


Fig. 8. CDF of request latency. (a) Normalized write latency. (b) Normalized read latency.

mapped to this type of active blocks, e.g., *smm*, the smaller improvement the write performance has.

B^{active}_{chb} has a small percentage in all applications except *lu*. This is because two-phase interleaving (i.e., $B^{active}_{approxPhases2}$) has better performance and thus we prioritize two-phase interleaving. Since all pages in $B^{active}_{allApprox}$ are written using fast approximate-write, a large percentage of $B^{active}_{allApprox}$, e.g, *fft*, tends to have better performance.

C. Read Response Time Comparison

Fig. 7 compares the read response time from different schemes, with the results normalized to `Baseline`. From the figure, `ApproxFTL` achieves 41.38% read response time reduction compared to `Baseline`. The improvement comes mainly from the reduction in read waiting time—approximate writes are faster so that the read operations wait shorter amount of time before being serviced.

The workload *mc* has negligible improvement because it only has a small read over write ratio, which means reducing write time has little impact on read waiting time.

Fig. 8 compares the cumulative distribution of different response time in servicing write and read requests, respectively, from workload *img* when adopting `ApproxFTL` and `Baseline`. We normalized the results to the worse write and read response time in `Baseline`, respectively. From the figure, more requests are serviced quicker in `ApproxFTL` than they are in `Baseline`. For example, the largest write

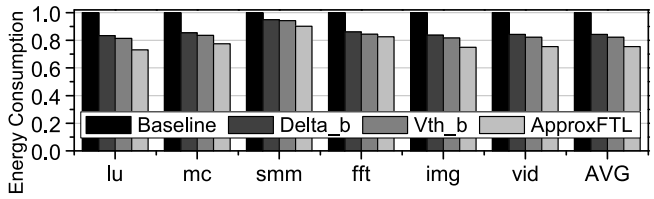


Fig. 9. Energy consumption comparison.

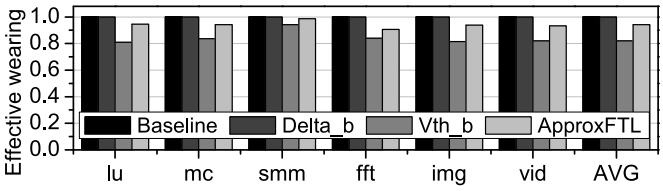


Fig. 10. Comparing chip lifetime.

response time in `ApproxFTL` is about 60.50% of that in `Baseline`.

Given an *x*-axis value, e.g., 0.269 or 26.9% of the longest latency in `Baseline`, we observed that about 61% (dot A) and 17% (dot B) of write requests are serviced with quicker than this time when using `ApproxFTL` and `Baseline`, respectively.

D. Energy Consumption Comparison

We next compared the energy consumption when adopting different schemes. We adopted the energy model from Hu *et al.* [53] to calculate the total energy consumption of the flash-based SSD and a typical 128 MB DRAM buffer inside the flash module. We model all three basic operations, read/write/erase in flash. Fig. 9 summarizes the normalized results over `Baseline`. From the figure, on average, `ApproxFTL` achieves 24.5%, 10.3%, and 8.2% improvements over `Baseline`, `Delta_b`, and `Vth_b`, respectively.

The large energy consumption reduction comes from two sources: 1) page reads and writes in `ApproxFTL` have shorter access latencies than they are in other schemes, and thus consume less energy and 2) `Vth_b` and `ApproxFTL` reduce the maximal threshold voltage when programming approximate pages such that each write consumes less energy than that in `Baseline`.

`ApproxFTL` achieves the greatest energy consumption reduction over `Baseline` in *lu*, i.e., 26.8%, and the smallest in *smm*, i.e., 9.9%.

E. Lifetime Comparison

We next compared the chip lifetime when adopting different schemes and summarized the results in Fig. 10. The results were normalized to `Baseline`. On average, `ApproxFTL` achieves about 5.75% chip lifetime improvement over the baseline.

`ApproxFTL` extends chip lifetime mainly because it uses a reduced erase voltage when erasing the blocks that only save approximate pages. The more $B^{used}_{approxPhase2}$ blocks the workload has, the large the lifetime improvement is. Since

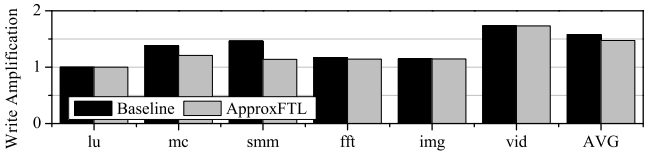


Fig. 11. Write amplification for each benchmark.

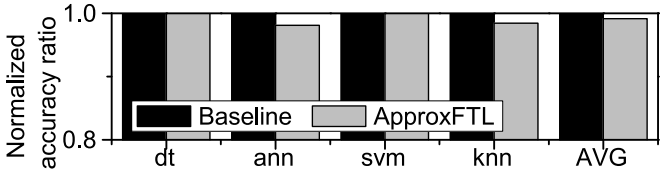


Fig. 12. Quality degradation for each benchmark.

all approximate pages in the Vth_b scheme are in separate blocks, it shows slightly better lifetime improvement than ApproxFTL.

F. Impact on Write Amplification

Approximate and precise data may be mixed in one block, e.g., B_{chb}^{active} or $B_{approxPhases2}^{active}$, we should evaluate the impact of the proposed FTL on the GC performance. In this experiment, the flash memory storage will be filled with uniform random workloads before each simulation, so that GC and write amplification may be triggered for evaluation, and we will provide evaluation on how different writing schemes affect the write amplification of the flash storage.

Fig. 11 compares the GC-induced write amplification in different schemes. Delta_b and Vth_b does not take hot/cold separation into consideration, and thus have the comparable write amplification quality as that in baseline. From the figure, ApproxFTL reduce the average write amplification by 6.64% over baseline. The write amplification reduction comes from the following.

1) Approximate data in scientific computing applications are more likely to be hot [54], and hot precise data will be allocated to B_{chb}^{active} or $B_{approxPhases2}^{active}$ blocks, thus we can expect that the victim hot GC block will be filled with invalid pages soon and it can be reclaimed with low overhead.
2) Separating hot write data into the hot pool in *img* and *vid* will largely reduce the overhead of GC.

G. Impact on Quality

In the last experiment, we evaluated the quality of the output. Due to the limitation of our simulator, we can only plug in error simulating approximate-write introduced data inaccuracy in a subset of applications.

Fig. 12 summarizes the normalized output quality from ApproxFTL. We used application-specific quality metrics that are widely in approximate computing research. Delta_b and Vth_b use the same RBER in relaxing approximate writes, and thus have the comparable output quality as that in ApproxFTL.

From the figure, we observed negligible reduction of output quality. There are three reasons as follows.

1) Occasional errors for noncritical data show negligible impact on quality loss, since these applications have the inherent error resilience.
2) In implementing the approximate write, ApproxFTL chooses conservative parameters such that approximate pages accumulates small number of errors, even we map precise pages to their neighbors.
3) We employ approximate page promotion policy so that the system prevents approximate pages from unbounded error-prone approximate writes.

VI. Conclusion

In this paper, we proposed ApproxFTL, an approximate-write aware FTL design for 3-D NAND flash memory storage systems. By reducing the maximal threshold voltage, we implement approximate write with significantly program disturbance reduction to neighboring pages.

Based on the tight correlation between reliability and performance, we devised pattern-guide page allocation design that prioritizes the interleaving of approximate and precise pages in LPN-to-PPN mapping. We then enhance wear leveling and GC policies in FTL to take advantage of the mitigated program disturbance in approximate write. Our experimental results show that ApproxFTL, while preserving high data quality, improves the read and write response time of flash accesses by 41.38% and 45.64% on average, respectively, and extends the lifetime of 3-D flash-based SSDs by 5.75% when comparing to the state-of-the-art.

Acknowledgment

The authors would like to thank the anonymous reviewers for their detailed and thoughtful feedback which improved the quality of this paper significantly. The authors would also like to thank A. Sampson for providing approximate applications on persistent storage.

References

[1] R. Micheloni, *3D Flash Memories*. Amsterdam, The Netherlands: Springer, 2016.
[2] T.-H. E. Yeh *et al.*, "Z-interference and Z-disturbance in vertical gate-type 3-D NAND," *IEEE Trans. Electron Devices*, vol. 63, no. 3, pp. 1047–1053, Mar. 2016.
[3] S. Aritome *et al.*, "Advanced DC-SF cell technology for 3-D NAND flash," *IEEE Trans. Electron Devices*, vol. 60, no. 4, pp. 1327–1333, Apr. 2013.
[4] C.-H. Hung *et al.*, "A highly scalable vertical gate (VG) 3D NAND flash with robust program disturb immunity using a novel PN diode decoding structure," in *Proc. Symp. VLSI Technol. (VLSIT)*, Honolulu, HI, USA, 2011, pp. 68–69.
[5] Y. Wang, Z. Shao, H. C. B. Chan, L. A. D. Bathen, and N. D. Dutt, "A reliability enhanced address mapping strategy for three-dimensional (3-D) NAND flash memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2402–2410, Nov. 2014.
[6] Y.-M. Chang, Y.-H. Chang, T.-W. Kuo, Y.-C. Li, and H.-P. Li, "Disturbance relaxation for 3D flash memory," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1467–1483, May 2016.
[7] Y.-M. Chang *et al.*, "On relaxing page program disturbance over 3D MLC flash memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 479–486.

[8] H.-S. Chang, Y.-H. Chang, T.-W. Kuo, Y.-M. Chang, and H.-P. Li, "A disturbance-aware sub-block design to improve reliability of 3D MLC flash memory," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Pittsburgh, PA, USA, 2016, pp. 1–10.

[9] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Trans. Comput. Syst.*, vol. 32, no. 3, p. 9, 2014.

[10] H. Tanaka *et al.*, "Bit cost scalable technology with punch and plug process for ultra high density flash memory," in *Proc. IEEE Symp. VLSI Technol.*, Kyoto, Japan, 2007, pp. 14–15.

[11] D. Kang *et al.*, "256 Gb 3 b/cell V-NAND flash memory with 48 stacked WL layers," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 210–217, Jan. 2017.

[12] W. Kim *et al.*, "Multi-layered vertical gate NAND flash overcoming stacking limit for terabit density storage," in *Proc. Symp. VLSI Technol.*, Honolulu, HI, USA, 2009, pp. 188–189.

[13] H.-T. Lue, S.-H. Chen, Y.-H. Shih, K.-Y. Hsieh, and C.-Y. Lu, "Overview of 3D NAND flash and progress of vertical gate (VG) architecture," in *Proc. IEEE 11th Int. Conf. Solid State Integr. Circuit Technol. (ICSICT)*, Xi'an, China, 2012, pp. 1–4.

[14] C.-C. Hsieh *et al.*, "Study of the interference and disturb mechanisms of split-page 3D vertical gate (VG) NAND flash and optimized programming algorithms for multi-level cell (MLC) storage," in *Proc. Symp. VLSI Technol. (VLSIT)*, Kyoto, Japan, 2013, pp. T156–T157.

[15] G. Wang *et al.*, "Low-cost low-power ASIC solution for both DAB+ and DAB audio decoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 913–921, Apr. 2014.

[16] H.-S. Chang, Y.-H. Chang, T.-W. Kuo, and H.-P. Li, "A light-weighted software-controlled cache for PCM-based main memory systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2015, pp. 22–29.

[17] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Asheville, NC, USA, 2013, pp. 123–130.

[18] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, May 2002.

[19] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *Proc. Conf. Design Autom. Test Europe*, Dresden, Germany, 2012, pp. 521–526.

[20] Y. Cai *et al.*, "Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, 2017, pp. 49–60.

[21] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 213–224, 2012.

[22] A. Sampson *et al.*, "EnerJ: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[23] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 301–312, 2012.

[24] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar, "High-density image storage using approximate memory cells," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 413–426, 2016.

[25] D. Jevdjic, K. Strauss, L. Ceze, and H. S. Malvar, "Approximate storage of compressed and encrypted videos," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 361–373.

[26] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Portland, OR, USA, 2015, pp. 554–566.

[27] H.-S. Lee, H.-S. Yun, and D.-H. Lee, "HFTL: Hybrid flash translation layer based on hot data identification for flash memory," *IEEE Trans. Consum. Electron.*, vol. 55, no. 4, pp. 2005–2011, Nov. 2009.

[28] M.-L. Chiao and D.-W. Chang, "ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 753–766, Jun. 2011.

[29] S. Lee, J. Park, K. Fleming, Arvind, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Trans. Consum. Electron.*, vol. 57, no. 4, pp. 1732–1739, Nov. 2011.

[30] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. FAST*, vol. 11. San Jose, CA, USA, 2011, pp. 77–90.

[31] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, 2009.

[32] X. Xu and H. H. Huang, "Exploring data-level error tolerance in high-performance solid-state drives," *IEEE Trans. Rel.*, vol. 64, no. 1, pp. 15–30, Mar. 2015.

[33] J. Jeong, S. S. Hahn, S. Lee, and J. Kim, "Improving NAND endurance by dynamic program and erase scaling," in *Proc. HotStorage*, San Jose, CA, USA, 2013, pp. 1–5.

[34] L. Shi *et al.*, "Retention trimming for lifetime improvement of flash memory storage systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 1, pp. 58–71, Jan. 2016.

[35] D.-S. Byeon *et al.*, "An 8 Gb multi-level NAND flash memory with 63 nm STI CMOS process technology," in *IEEE Int. Solid State Circuits Conf. Dig. Tech. Papers (ISSCC)*, San Francisco, CA, USA, 2005, pp. 46–47.

[36] Y. Pan, G. Dong, and T. Zhang, "Exploiting memory device wear-out dynamics to improve NAND flash memory system performance," in *Proc. FAST*, vol. 11. San Jose, CA, USA, 2011, pp. 245–258.

[37] N. Mielke *et al.*, "Bit error rate in NAND flash memories," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Phoenix, AZ, USA, 2008, pp. 9–19.

[38] JEDEC Standard, *Stress-Test-Driven Qualification of Integrated Circuits*, JEDEC Standard JESD47J.01, 2007, pp. 1–26.

[39] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *Proc. HotStorage*, 2014, pp. 1–5.

[40] L. M. Grupp, J. D. Davis, and S. Swanson, "The harey tortoise: Managing heterogeneous write performance in SSDs," in *Proc. USENIX Annu. Tech. Conf.*, San Jose, CA, USA, 2013, pp. 79–90.

[41] L. Shi *et al.*, "Exploiting process variation for write performance improvement on NAND flash memory storage systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 334–337, Jan. 2016.

[42] G. Yadgar, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving SSD erase costs using WOM codes," in *Proc. FAST*, Santa Clara, CA, USA, 2015, pp. 257–271.

[43] S. Im and D. Shin, "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," *J. Syst. Archit.*, vol. 56, no. 12, pp. 641–653, 2010.

[44] L.-P. Chang, "Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs," in *Proc. Asia South Pac. Design Autom. Conf. (ASPDAC)*, Seoul, South Korea, 2008, pp. 428–433.

[45] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: Improving NAND flash lifetime by balancing page endurance," in *Proc. FAST*, vol. 14. Santa Clara, CA, USA, 2014, pp. 47–59.

[46] S. Odeh and Y. Cassuto, "NAND flash architectures reducing write amplification through multi-write codes," in *Proc. 30th Symp. Mass Storage Syst. Technol. (MSST)*, Santa Clara, CA, USA, 2014, pp. 1–10.

[47] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *Proc. 31st Symp. Mass Stor. Syst. Technol. (MSST)*, Santa Clara, CA, USA, 2015, pp. 1–14.

[48] R. Chen, Y. Wang, D. Liu, Z. Shao, and S. Jiang, "Heating dispersal for self-healing NAND flash memory," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 361–367, Feb. 2017.

[49] R. Chen *et al.*, "Image-content-aware i/o optimization for mobile virtualization," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 1, p. 12, 2016.

[50] R. Chen, Y. Wang, and Z. Shao, "Dheating: Dispersed heating repair for self-healing NAND flash memory," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Montreal, QC, Canada, 2013, pp. 1–10.

[51] Y. Hu *et al.*, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. Int. Conf. Supercomput.*, Tucson, AZ, USA, 2011, pp. 96–107.

[52] J.-W. Im *et al.*, "7.2 a 128Gb 3b/cell V-NAND flash memory with 1Gb/s I/O rate," in *Proc. IEEE Int. Solid State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2015, pp. 1–3.

[53] Y. Hu *et al.*, "Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation," in *Proc. IEEE 26th Symp. Mass Stor. Syst. Technol. (MSST)*, 2010, pp. 1–12.

[54] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th Annu. Design Autom. Conf.*, Austin, TX, USA, 2013, pp. 113–121.

Jinhua Cui received the B.S. degree from Southwest University, Chongqing, China, in 2012. She is currently pursuing the Ph.D. degree with Xi'an Jiaotong University, Xi'an, China, under the guidance of Prof. W. Wu.

She was a visiting student with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA, USA, from 2016 to 2017, co-advised by Prof. Y. Zhang and Prof. J. Yang. Her current research interests include NAND flash memory-based storage system, approximate storage, big data, cloud computing, HDFS, and database index.

Youtao Zhang (M'17) received the B.S. and M.E. degrees from Nanjing University, Nanjing, China, in 1993 and 1996, respectively, and the Ph.D. degree in computer science from the University of Arizona, Tucson, AZ, USA, in 2002.

He is currently an Associate Professor of computer science with the University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include computer architecture, program analysis, optimization, on-chip interconnection, architectural support for security, new memory technologies, and networks-on-chip.

Prof. Zhang was a recipient of the U.S. National Science Foundation Career Award in 2005. He is a member of ACM.

Liang Shi (M'17) received the B.S. degree in computer science from the Xi'an University of Post and Telecommunication, Xi'an, China, in 2008 and the Ph.D. degree from the University of Science and Technology of China, Hefei, China, in 2013.

He is currently an Associate Professor with the College of Computer Science, Chongqing University, Chongging, China. His current research interests include flash memory, embedded systems, and emerging nonvolatile memory technology.

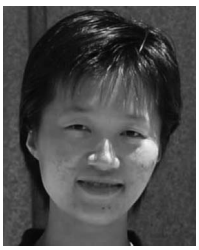
Chun Jason Xue (M'17) received the B.S. degree in computer science and engineering from the University of Texas at Arlington, Arlington, TX, USA, in 1997 and the M.S. and Ph.D. degrees in computer Science from the University of Texas at Dallas, Richardson, TX, USA, in 2002 and 2007, respectively.

He is currently an Assistant Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His current research interests include memory and parallelism optimization for embedded systems, software/hardware co-design, real time systems, and computer security.

Weiguo Wu received the B.S., M.S., and Ph.D. degrees in computer science from Xi'an Jiaotong University, Xi'an, China, in 1986, 1993, and 2006, respectively.

He is currently a Professor with the School of Electronic and Information Engineering, Xi'an Jiaotong University, where he is also the Deputy Director of the Neo Computer Institute. His current research interests include high performance computing, computer network, embedded system, VHDL, cloud computing, and flash memory.

Dr. Wu is a Senior Member of Chinese Computer Federation, a Standing Committee Member of High Performance Computing Clusters in Chinese Computer Federation and Microcomputer (Embedded System) in Chinese Computer Federation, and the Director of Shaanxi Computer Federation.



Jun Yang received the B.S. degree in computer science from Nanjing University, Nanjing, China, in 1995, the M.A. degree in mathematical sciences from Worcester Polytechnic Institute, Worcester, MA, USA, in 1997, and the Ph.D. degree in computer science from the University of Arizona, Tucson, AZ, USA, in 2002.

She is a Professor with the Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA, USA. Her current research interests include low power and temperature-aware micro-architecture designs, new memory technologies, and networks-on-chip.

Dr. Yang was a recipient of the U.S. NSF Career Award in 2008.