RiF: Improving Read Performance of Modern SSDs Using an On-Die Early-Retry Engine

Myoungjun Chun*[1], Jaeyong Lee*[1], Myungsuk Kim[2], Jisung Park[3], Jihong Kim[1]
[1]Seoul National University, [2]Kyungpook National University, [3]POSTECH
[1]{mjchun, jylee, jihong}@davinci.snu.ac.kr, [2]ms.kim@knu.ac.kr, [3]jisung.park@postech.ac.kr

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00056

***Abstract*—Modern high-performance SSDs have multiple flash channels operating in parallel to achieve their high I/O bandwidth. However, when the effective bandwidth of these flash channels declines, the SSD's overall bandwidth is substantially impacted. In contemporary SSDs featuring high-density 3D NAND flash memory, frequent invocations of a read-retry procedure pose a significant challenge to fully utilizing the maximum I/O bandwidth of a flash channel. In this paper, we propose a novel read-retry optimization scheme, Retry-in-Flash (RiF), which proactively minimizes the amount of time wasted in conventional read-retry procedures. Unlike existing read-retry solutions that focus on identifying an optimal read-reference voltage for a sensed page, the RiF scheme focuses on determining early on whether a read-retry will be required for the sensed data. To know if a read-retry is needed or not at the earliest possible time, we propose a RiF-enabled flash chip with an on-die early-retry (ODEAR) engine. When the ODEAR engine determines that a sensed page requires a read-retry, a read-reference voltage is immediately adjusted and the same page is re-read while ignoring the previously sensed page. By performing the key steps of a read-retry procedure inside a RiF flash chip without transferring the sensed uncorrectable page to an off-chip controller, the RiF scheme prevents the read bandwidth of a flash channel from being wasted due to failed read data. To evaluate the RiF scheme, we developed a prototype RiF-enabled flash chip and constructed a RiF-aware SSD simulator using RiF flash chips. Our evaluation results show that the proposed RiF scheme improves the effective SSD bandwidth by 72.1% on average over a state-of-the-art read-retry solution at 2K P/E cycles with negligible power and area overheads.**

I. Introduction

Modern solid-state drives (SSDs) that employ high-density 3D NAND flash memory are widely used to meet the ever-increasing I/O performance and storage capacity requirements of data-intensive applications such as graph analytics [1]–[4] and machine learning-based applications [5], [6]. By parallelizing I/O operations across multiple high-speed flash channels, each supporting multiple NAND flash dies, a modern SSD can provide both high I/O performance and high capacity. For example, a recent 3.84-TB SSD [7], capable of reaching a 6.9-GB/s sequential read bandwidth, utilizes eight flash channels, with each channel supporting a 480-GB capacity. To facilitate parallel operation of multiple flash channels, each channel is equipped with its own dedicated hardware engine for supporting strong error-correcting codes (ECC) [8]–[13].

In order for high-performance SSDs to achieve their maximum SSD I/O bandwidth *from a host side*, it is important that almost 100% of the I/O bandwidth of each flash channel should be used for serving host I/O requests without redundant or wasted channel utilization. That is, the effective I/O bandwidth of a flash channel for host I/O requests should be near the maximum I/O bandwidth of the flash channel. When the effective I/O bandwidth of a flash channel deteriorates, the overall effective SSD I/O bandwidth is directly affected. For example, when 30% of the I/O bandwidth of a flash channel is used for SSD-internal data movements for garbage collection and read-disturb management [14]–[17], the overall SSD I/O bandwidth cannot be higher than 70% of its peak I/O bandwidth.

In contemporary high-capacity SSDs with high-density 3D NAND flash memory, a read-retry procedure [18]–[23], which is essential for supporting reliable reads, can be a primary source for posing a significant challenge to fully utilizing the maximum I/O bandwidth of a flash channel by introducing a large amount of SSD-internal reads. Due to error-prone characteristics of high-density 3D NAND flash memory [20], [21], [24]–[26], even strong ECC engines (e.g., 72 bits per 1-KiB codeword [10]) of high-performance SSDs often fail to completely eliminate uncorrectable errors from a read page. To address this issue, when the raw bit error rate (RBER) of a read page exceeds the ECC correction capability (i.e., the maximum number of raw bit errors that an ECC engine can correct), a modern SSD initiates a read-retry procedure. A conventional read-retry procedure involves iterating through a sequence of three read-retry steps, rs_1, rs_2, and rs_3, until the RBER of the read page falls below the ECC correction threshold. In step rs_1, the read-reference voltage (V_{REF}) is adjusted; in step rs_2, the same page is re-read using the adjusted V_{REF}; and in step rs_3, the read page is decoded by an ECC engine.

While a read-retry procedure is crucial for maintaining the reliability of modern NAND flash memory, invoking a read-retry procedure amplifies a single host read by (1 + N_{RR}) times, where N_{RR} represents the total number of repeated read-retry sequences. As the same page needs to be read (1 + N_{RR}) times, the effective read bandwidth of a flash channel may be significantly reduced when a substantial number of host reads necessitate the read-retry procedure. To alleviate the performance penalty associated with read-retry procedures, many studies [19], [22], [23], [27]–[31] have proposed various solutions aimed at reducing N_{RR} by selecting an appropriate V_{REF} value, which enables successful decoding of a read page by the ECC engine. For example, a read-retry mitigation tech-

*Both authors contributed equally to this work.

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00056

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

nique [23] predicts near-optimal V_{REF} values for a previously failed page using spare cells as error indicators. This approach can decrease the average N_{RR} value to 1.2.

Although existing read-retry solutions have been relatively successful in reducing N_{RR}, their efficiency is inherently constrained due to their reactive nature: The read-retry procedure is initiated *after* it is determined that a read page cannot be decoded by an *off-chip* ECC engine. When a read-retry procedure was rarely invoked as in 2D NAND flash memory, the existing reactive off-chip approach was an efficient solution for handling a read-retry request. However, we have observed that the same approach is not adequate for modern high-density 3D NAND flash memory because a read-retry procedure is much more frequently invoked. (See Section III-A.)

We identified three key weaknesses of existing read-retry solutions when they were applied in high-density 3D NAND flash memory. First, a decision to invoke a read-retry procedure is made by an *off-chip* ECC engine. Since an ECC-failed initial page must be sent to the off-chip ECC engine to activate the read-retry procedure, at least two reads are required even with an ideal read-retry solution (that reduces N_{RR} to 1), resulting in the wasted read bandwidth of a flash channel for the first failed read page. Second, a decision to invoke a read-retry procedure is made *too late*. The read-retry decision is made reactively, based on the number of uncorrectable raw bit errors after the ECC decoding step (i.e., rs_3). This requires the sensed page to be transferred to the (off-chip) ECC engine. When the sensed page cannot be corrected by the ECC engine, the read bandwidth of the flash channel is wasted. Third, large variation in ECC decoding latency often decreases the effective channel bandwidth by delaying the subsequent read transfer through a flash channel. The ECC decoding latency of the first failed pages tends to be much longer than that of the successfully decoded pages. During this extended decoding interval for the failed read page, the corresponding flash channel cannot accommodate any other requests and must remain idle, wasting the read bandwidth of the flash channel.

In this paper, we propose a novel read-retry optimization scheme, Retry-in-Flash (RiF), which overcomes the key limitations of the existing read-retry solutions for modern 3D NAND flash memory where an invocation of a read-retry procedure is a common case for most reads. The key novelty of the RiF scheme comes from 1) decoupling a decision for a read-retry from an ECC decoding procedure and 2) making the read-retry decision at the earliest possible time. To support key features of the RiF scheme, we propose a RiF-enabled flash chip with an On-Die EArly-Retry (ODEAR) engine. The ODEAR engine consists of two main modules: the read-retry predictor (RP) module and the read-voltage selector (RVS) module. When a page is sensed, the RP module estimates if the sensed page can be correctly decoded by an off-chip ECC engine or not. When the RP module predicts that a read-retry would be necessary, the RVS module chooses the next V_{REF} values. Note that both RP and RVS modules work on-die without any assistance from an off-chip controller. By performing a read-retry sequence within a flash die, when a read page necessitates a read-retry, the ODEAR engine proactively inhibits unnecessary off-chip transfer of the read page and its long wasted ECC decoding, thus minimizing the wasted flash channel bandwidth caused by the read-retry.

There are two key challenges in designing and implementing the ODEAR engine. First, the RP module of the ODEAR engine should have a sufficiently high prediction accuracy for read-retries. For a read-retry prediction, the RP module exploits a strong correlation between the RBER value of a sensed page and the syndrome weight of the sensed page in its LDPC decoding. We validated that the RP module can achieve 98.7% prediction accuracy on the correctability of a sensed page, effectively reducing the majority of uncorrectable off-chip data transfers. Second, the ODEAR engine should have no overhead on existing flash dies from the power/performance/area (PPA) perspective. For a die-level efficient implementation with a minimal PPA impact, the proposed RP module focuses on predicting if a read retry is necessary or not for a read page without decoding the read page. We validated that the PPA impact of the RP module is minimal from a synthesis result using a commercial EDA tool.

To evaluate the effectiveness of the RiF scheme at the SSD level, we developed a RiF-aware SSD simulator, RiFSSD, and compared its performance with ones with the state-of-art read-retry solutions, Sentinel [23] and Swift-Read [32]. Our evaluation results using eight real-world workloads showed that RiFSSD enhances the I/O bandwidth significantly when compared with the state-of-the-art off-chip-based technique [23]. The improvements amounted to 23.8%, 47.4%, and 72.1% on average at 0K, 1K, and 2K P/E cycles, respectively.

II. Background

In this section, we review the basics of NAND flash memory and explain reliability problems encountered in modern SSDs with their mitigation techniques.

A. Overview of NAND Flash Memory

1) Basic Operations: NAND flash memory employs a unique type of transistor, called a flash cell, which can change its threshold voltage (V_{TH}) by injecting or ejecting electrons into the SiN layer. By varying the number of electrons stored in the flash cell, the flash cell can distinguish different V_{TH} states which are used to represent different digital states. Read/write operations are performed at the unit of a *page* (e.g., a 16-KiB page), while erase operations are performed at the unit of a *block* (e.g., a 9-MiB block) that consists of a set of pages.

Modern NAND flash memory adopts multi-level cell (MLC) techniques that store m-bit information in a single flash cell by using 2^m distinct V_{TH} states. Fig. 1(a) illustrates V_{TH} distributions for 4-state NAND flash memory, which stores two bits per cell. The stored data in a flash cell can be read by identifying the V_{TH} state of the flash cells using the read-reference voltages V_{REF}^{Ri}'s. For example, to read an LSB page, the flash chip applies the V_{REF}^{R2} voltage to identify if the V_{TH} of the flash cell is P2 state or higher.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

2) Read Errors in NAND Flash: In an ideal scenario, the V_{TH} of programmed flash cells remains stable, ensuring reliable data reading. However, as the flash cells repeatedly experience program and erase operations (called P/E cycles), the dielectric layer known as T_{OX}, responsible for electrically isolating the SiN layer from the channel, is gradually damaged. As the damage to T_{OX} is accumulated, unexpected traps (i.e., defects) are generated in T_{OX} [33]. These traps make T_{OX} an inefficient insulator, resulting in charge leakage paths from/to the SiN layer. Consequently, the V_{TH} distribution of flash cells gets widened and/or shifted, primarily due to charge leakage occurring under various noise conditions such as long retention time, repetitive read operations [21], [34]–[36]. Moreover, in 3D NAND flash memory, flash cells can experience substantial V_{TH} shifts even in their early lifetimes from lateral charge diffusion [37], [38]. Such V_{TH} distortions cause flash cells to deviate from their original V_{REF}^{Ri}, resulting in read errors, as illustrated in Fig. 1(b). In recent triple-level cell (TLC) NAND flash memory, due to the narrower margin between V_{TH} states, the number of bit errors easily exceeds ECC correction capability after only a few weeks of the data retention time. (Section III-A.)

B. Error Correction in SSDs

To ensure reliable reads of stored data from error-prone flash cells, modern SSDs employ strong error correction schemes. Among many error correction schemes (e.g., BCH, Reed-Solomon), low-density parity check (LDPC) codes are widely used for modern 3D TLC NAND flash memory because of their high error correction capability. When a sensed page is correctly decoded by the LDPC engine, the read operation is completed. However, when the RBER of the sensed page exceeds the correction capability of the LDPC engine, a read-retry procedure is repeated until the LDPC engine successfully decodes the sensed page, or until the sequence has been repeated up to a predetermined maximum number of times.

1) Overview of LDPC Code: The encoding and decoding scheme of LDPC is determined by a parity check matrix, denoted as $\boldsymbol{H}$, where the width is equal to the length of a codeword and the height is equal to the number of parity bits [39], [40]. Fig. 2 shows an example parity check matrix $\boldsymbol{H}$ where the width and height are 8 and 4, respectively. In each row of $\boldsymbol{H}$, 1's represent the bit positions of a codeword $\boldsymbol{c} = (c_0, c_1, .., c_7)$ that are used for computing a syndrome vector $\boldsymbol{S} = (s_0, s_1, s_2, s_3)$, which enables the error detection/correction of codewords. In our example, the first row

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{S}^T = \boldsymbol{H}\boldsymbol{c}^T = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} c_1 \oplus c_3 \oplus c_4 \oplus c_7 \\ c_0 \oplus c_1 \oplus c_2 \oplus c_5 \\ c_2 \oplus c_5 \oplus c_6 \oplus c_7 \\ c_0 \oplus c_3 \oplus c_4 \oplus c_6 \end{bmatrix}$$

M = 4, N = 8

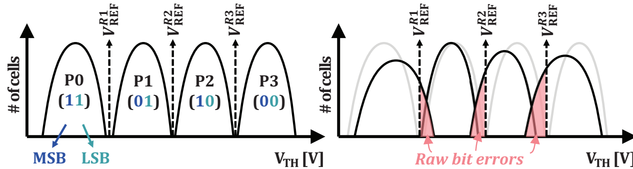
Fig. 2: An example parity check matrix $\boldsymbol{H}$ with its syndrome vector $\boldsymbol{S}$ for a codeword $\boldsymbol{c}$.

of $\boldsymbol{H}$ represents a parity check equation, $c_1 + c_3 + c_4 + c_7 = 0$, for a codeword $\boldsymbol{c}$. We compute s_0 by performing an XOR operation on c_1, c_3, c_4 and c_7.

The LDPC encoder generates a codeword by adding parity bits next to the data bits so that all parity check equations are satisfied. Therefore, if no errors occur, the Hamming weight of syndromes (i.e., $\sum s_k$) must be 0. If errors occur in a sensed page, the LDPC decoder initially calculates the probability of each bit originally being 0 or 1, based on syndrome bits. Subsequently, the decoder updates each bit of the original sensed page to the value with a higher probability. After that, the decoder verifies the integrity of updated data by checking whether the syndrome weight is 0 or not. If the syndrome weight is 0, the decoder considers the page error-free and completes the decoding process. If not, the decoder executes the above procedure iteratively until the syndrome weight becomes 0. As the RBER of a sensed page increases, more iterations are required to correct the errors, which leads to an increase in the decoding latency of LDPC [12], [13], [39], [41]. If the data cannot be corrected after a preset maximum number of iterations (e.g., 20), the LDPC decoder considers the data uncorrectable and declares a decoding failure.

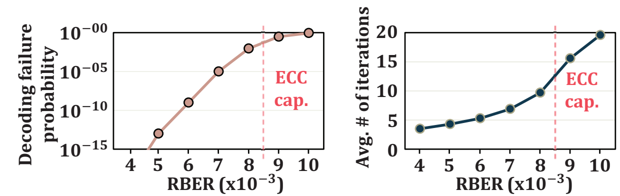
Figs. 3(a) and 3(b) show simulation results of the error correction capability of the 4-KiB quasi-cyclic LDPC (QC-LDPC [11]–[13]) decoder which is commonly adopted in modern SSDs due to its ease of hardware implementation. As the RBER increases, the likelihood of decoding failure as well as the number of necessary decoding iterations generally rises. Particularly, when the RBER surpasses 0.0085, the probability of a decoding failure exceeds 10^{-1} while the number of required iterations reaches 20.

2) V_{REF} Adjustment Techniques: When an LDPC engine fails to decode the sensed page, the flash controller adjusts V_{REF}^{Ri} values (i.e., a read-retry procedure) so that the number of raw bit errors from the subsequently sensed page can be reduced below the correction capability of the LDPC engine. To facilitate this process, flash manufacturers often employ



(a) An ideal V_{TH} distribution. (b) A distorted V_{TH} distribution.

Fig. 1: Changes in V_{TH} distributions under noisy conditions.



(a) Decoding failure probability. (b) Average no. of iterations.

Fig. 3: Error correction capability of a 4-KiB QC-LDPC.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

a predetermined sequence of V_{REF}^{Ri} values. When the current V_{REF}^{Ri} values fail to decode the sensed page, the next V_{REF}^{Ri} values in the sequence (following the current V_{REF}^{Ri} values) are selected for re-sensing the failed page. As the density of flash memory increases, due to the narrower V_{TH} distributions, read-retries occur more frequently and often necessitate multiple iterations along the V_{REF}^{Ri} sequence, which results in a decline in read performance. In general, the page-read latency tREAD can be formulated as follows:

$$\texttt{tREAD} = (\texttt{tR} + \texttt{tDMA} + \texttt{tECC}) \times (N_{RR} + 1) \qquad (1)$$

where tR, tDMA, and tECC are the latencies of sensing the page data, transferring the sensed data from the chip to the SSD controller, and decoding the data with the ECC engine, respectively. To mitigate the performance penalty associated with multiple read-retries, it is essential to reduce N_{RR} by choosing better V_{REF}^{Ri} values that successfully correct the previously failed page.

III. Read Retries in Modern 3D-Flash SSDs

A. Frequency of Read-Retry Invocations

To understand how frequently a read-retry procedure is invoked in a modern NAND flash memory, we conducted a comprehensive study using 160 real 3D TLC NAND flash chips under varying operating conditions. Since a read-retry procedure is initiated when the RBER value of a page exceeds the ECC correction capability (i.e., the RBER threshold that an off-chip ECC engine can tolerate), we measured the RBER value from more than 10^5 pages for each operating conditions, which were randomly chosen from the 160 flash chips.

Fig. 4 shows the distributions of required retention time until the RBER value of a page exceeds the ECC correction capability for six P/E-cycle counts in the flash lifetime. A box at (x, y) represents the proportion of pages in which the RBER value of a page exceeds the RBER threshold after x-days of retention at y P/E cycles. As shown in Fig. 4, we observed that the RBER value of a page increases beyond the ECC correction capability in short retention times. Under 0 P/E cycle, 200 P/E cycles, and 500 P/E cycles, a read-retry procedure may be invoked after the data retention time of 17 days, 14 days, and 10 days, respectively. Considering that most commercial SSDs are required to support longer than 4-week retention time, our results strongly indicate that a read-retry procedure can be invoked for most read requests. For example, the JEDEC industry specification [42] requires that enterprise SSDs should support at least 3-month data

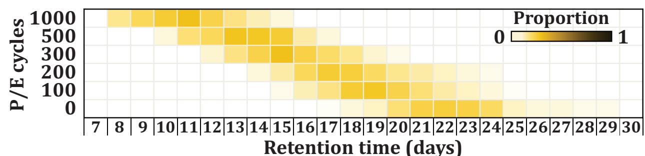


Fig. 4: Distributions of required retention time until the RBER value of a page exceeds the ECC correction capability under different operating conditions.

retention time. Judging by the results in Fig. 4, when an SSD experiences about 1K P/E cycles, read requests to pages after just 8-day retention time suffer performance degradation from read-retry procedures. Furthermore, note that due to the error-prone nature of modern 3D NAND flash memory, the read-retry procedure is required even in a fresh wear-out condition (as under 0 P/E cycle). The results summarized in Fig. 4 strongly suggest that an invocation of a read-retry procedure is a common-case event that affects most read requests.

B. Limitations of Existing Read-Retry Solutions

In order to mitigate the overhead of frequent read-retry procedures, many prior investigations [19], [22], [23], [27]–[31], [43], [44] have focused on reducing N_{RR} which is equal to the total number of failed ECC decoding of a sensed page. Among many existing off-chip-based read-retry optimization techniques, two recent works, Sentinel [23] and Swift-Read [45], are quite successful in reducing N_{RR}. For example, Sentinel reduced the average number of failed ECC decoding to 1.2 [23].

Sentinel [23] stores predefined bit patterns in spare cells (called Sentinel Cells) for each page and estimates near-optimal V_{REF} values based on errors to the Sentinel Cells of the page. Depending on a target page's type (e.g., LSB/CSB/MSB pages in TLC NAND flash memory), reading the page's Sentinel Cells may need to use different V_{REF} values from ones used for the first failed page. In such a case, a read-retry procedure in Sentinel must perform two off-chip reads even when it can accurately predict near-optimal V_{REF} values. That is, for each failed read in Sentinel, an extra off-chip read to retrieve Sentinel Cells from the target page may be needed, thus, increasing the number of off-chip reads up to twice the value of N_{RR}.

Swift-Read [32] performs a read-retry procedure via a single NAND flash command, which performs two reads to the target page inside the chip; the first read uses a predefined V_{REF} value[1] for the near-optimal V_{REF} prediction heuristic while the second read uses optimized V_{REF} values based on the number of one's in the first read. Since the number of one's tends to be uniformly distributed in a WL (regardless of its location) because of a data randomization step [9], [46]–[48] when a page is programmed, the Swift-Read scheme can accurately determine the optimal V_{REF} values using the difference between the measured number of one's over the expected number of one's when no bit errors exist.

In order to understand the maximum efficiency of the state-of-art off-chip read-retry solutions (such as Sentinel and Swift-Read) in high-performance SSDs based on modern 3D NAND flash memory, we measured the I/O bandwidth of SSD_{one}, an SSD with an ideal read-retry solution (i.e., N_{RR} = 1 for retried reads), using an NVMe SSD simulator. (For a more detailed description of our simulation environment, see Section VI.) To compare the performance of SSD_{one} with ideal performance,

[1] The most representative V_{REF} value for near-optimal V_{REF} prediction, as determined by manufacturers after extensive profiling of NAND flash chips.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

we built SSD$_{\text{zero}}$, an SSD with no retried reads. That is, SSD$_{\text{zero}}$ always succeeds in its ECC decoding of a sensed page.

1) Evaluation Setup: To evaluate the I/O bandwidth of two SSDs, SSD$_{\text{one}}$ and SSD$_{\text{zero}}$, we extended MQSim [49], a state-of-the-art simulator for modern SSDs, so that it can support a read-retry procedure in its read path. In the extended MQSim, we added two key modifications for faithfully modeling an existing read-retry procedure. First, we modified the NAND flash model of MQSim such that the reliability characteristics of each simulated page follow that of real flash pages based on the real-device characterization of 160 3D TLC NAND flash chips. Second, when simulating a read request to a page, the extended MQSim mimics the latency for decoding the target page and invokes a read-retry procedure when the page's RBER exceeds the ECC correction capability.

To accurately reflect the characteristics of modern high-performance SSDs, our SSD models closely follow the key features of modern high-performance SSDs. Fig. 5 depicts the organization of the target SSD and its key parameters. We set the host interface for this evaluation to 8.0 GB/s [50]. To fully support the peak bandwidth of the host interface, we configure the target SSD with 8 channels, each supporting a peak bandwidth of 1.2 GB/s (1.2 GB/s $\times$ 8 $>$ 8.0 GB/s). The channel and a channel-level ECC decoder (which is a typical architecture of modern SSDs [49]) are shared by four 4-plane dies.[2] We set the average page-read latency of 40 μs [45] and assume that the latency of the ECC decoder varies from 1 μs to 20 μs depending on the target page's RBER [12], [13].

2) Performance Evaluation: Fig. 6 shows how the I/O bandwidth of SSD$_{\text{one}}$ changes across three distinct stages of SSD life cycle (0K, 1K, and 2K P/E cycles) under four real-world workloads [51]. (For a detailed description of the four

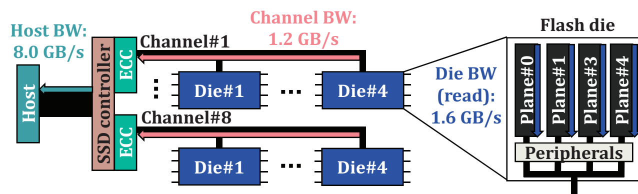


Fig. 5: An overall architecture of the target high-performance SSD.

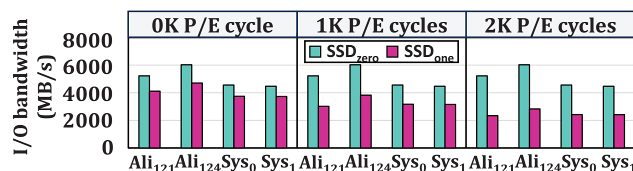


Fig. 6: A comparison of I/O bandwidth between SSD$_{\text{one}}$ and SSD$_{\text{zero}}$.

[2]When a page is read, the data path between a channel-level ECC decoder and a flash channel is dedicated. That is, the flash controller should first transfer the sensed (raw) page from the flash chip to the channel-level ECC decoder (via a flash channel) in order to correct any potential bit errors of the target page.

workloads, see Section VI-A.) We make two observations from the results. First, SSD$_{\text{one}}$ is unable to prevent the substantial performance drop caused by frequent read-retry procedures. The I/O bandwidth of the SSD degrades by 19.4%, 34.9%, and 50.4% on average compared to when no read-retry procedure occurs in the 0K, 1K, and 2K P/E cycles, respectively. Second, read performance is typically a crucial requirement for most applications, but the I/O bandwidth of SSD$_{\text{one}}$ experiences more significant degradation in read-intensive workloads. For instance, under the most read-intensive workload Ali$_{124}$, the bandwidth of SSD$_{\text{one}}$ is limited to at most 2831 MB/s at 2K P/E cycles, whereas SSD$_{\text{zero}}$ successfully sustains the 6026 MB/s I/O bandwidth demanded by the host interface. Our evaluation results highlight the fundamental limitations of existing off-chip read-retry solutions that have primarily focused on minimizing N_{RR} by efficiently determining near-optimal V$_{\text{REF}}$. Although these techniques effectively reduce N_{RR} to a minimum of one, they still fall short of achieving the optimal performance of an SSD under all operating conditions.

3) Root Cause Analysis: In order to more comprehensively understand the limitations of existing off-chip read-retry solutions, we examined the execution timeline of a read request in two SSDs, SSD$_{\text{zero}}$ and SSD$_{\text{one}}$. This detailed examination enables us to identify performance bottlenecks and inefficiencies in the read-retry procedure in SSD$_{\text{one}}$. Figs. 7(a) and 7(b) illustrate the key differences in the execution timeline to complete a 256-KiB sequential read request from a host, which is divided into four 64-KiB multi-plane read commands, A, B, C, and D, in SSD$_{\text{zero}}$ and SSD$_{\text{one}}$, respectively. For easier comparison, we assumed that a flash channel and an off-chip ECC engine are shared by two flash dies, which consist of 4 planes (i.e., 8 planes in total). We assumed that A and B require read-retry procedures to ensure data reliability in SSD$_{\text{one}}$. Since all planes of a flash die can operate simultaneously, each die can complete a 64-KiB multi-plane read command (i.e., 16-KiB pages $\times$ 4 planes) in the flash read latency `tR`. As a result, when two dies of a flash channel are fully operational, a total of 128-KiB of data can be sensed within a single `tR` interval. Once two multi-plane read commands, A and B, are completed, the 128-KiB data can be transferred to the SSD controller (after `tDMA`) and decoded by the off-chip ECC engine (after `tECC`). With a flash channel bandwidth of 1.2 GB/s, each die requires `tDMA` = 53 μs to transfer 64-KiB of data to the SSD controller. To achieve a high I/O throughput, the target SSD issues multi-plane read commands C and D to the target flash dies while the previously read data is being

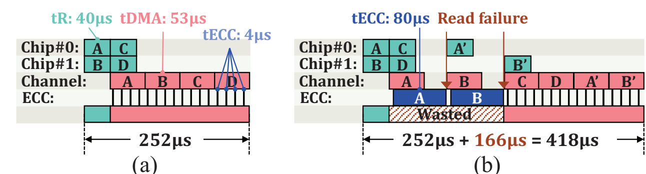


Fig. 7: A comparison of the execution timeline of a 256-KiB read request in SSD$_{\text{zero}}$ and SSD$_{\text{one}}$.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

transferred through the channel.

From our root cause study, we observed three key factors that degrade the efficiency of existing off-chip read-retry solutions. First, since an off-chip ECC engine decides to initiate a read-retry procedure, it is not known whether a read-retry is necessary until the read data is decoded. As a result, at least one round of unnecessary `tR`, `tDMA`, and `tECC` is needed to start the read-retry procedure. As shown in Fig. 7(b), read-retry procedures for A and B (denoted by A' and B') are initiated only after A and B were declared as decoding failures at the off-chip ECC engine. The failed read commands, A and B, therefore, delay the processing of subsequent read commands, C and D, which do not require a read-retry procedure. This delay results in an increase in the total execution time of SSD_{one} by 166 μs over SSD_{zero}.

Second, since both unrecoverable pages, as well as decodable pages (by the off-chip ECC engine), should be transferred to the off-chip ECC engine, the effective channel bandwidth of SSD_{one} is significantly degraded. The effective channel bandwidth plays a crucial role in fully utilizing the I/O bandwidth potential of high-performance SSDs. For example, in our target SSD shown in Fig. 5, a single flash die capable of providing up to 1.6 GB/s (equivalent to reading 64-KiB data per 40 μs with 16-KiB $\times$ 4 planes) could saturate a flash channel with a 1.2 GB/s bandwidth. Unfortunately, the existing off-chip read retry solutions are unable to prevent uncorrectable pages from consuming the channel bandwidth. As shown in Fig. 7(b), the two failed read commands, A and B, result in a 33.3% reduction in the effective channel bandwidth of SSD_{one}, while SSD_{zero} fully utilizes the channel bandwidth for ECC-decodable pages.

Third, we observed that the channel bandwidth can be less efficiently utilized when the ECC decoding latency (i.e., `tECC`) is increased. Depending on the page's RBER value, the decoding latency can vary significantly up to 20 times (as shown in Fig. 3(b)). When an uncorrectable page is decoded by an ECC engine, its `tECC` is much longer than that of an ECC-decodable page. For example, in Fig. 7(b), four unrecoverable pages, which were read by the read commands A, take 80 μs for their failed decoding. During this period, the channel bandwidth is completely wasted since the limited-capacity buffer of the channel-level ECC decoder becomes full, preventing it from accepting more raw data from the next sensed page (e.g., C and D cannot be transferred while A is being decoded).

IV. Design of RiF Scheme

The limitations of existing read-retry solutions come from inefficient data movement of uncorrectable pages to an off-chip ECC decoder. To overcome the key weakness of the existing solutions, we incorporate the concept of near-data processing into the read-retry mechanism. By enabling flash chips to decide if a sensed page can be correctable by an off-chip ECC decoder, we can mitigate the read-retry overhead by avoiding uncorrectable off-chip data transfers. To emphasize that read-retry processing is supported at the flash die level inside a flash chip, we call our proposed scheme, Retry-in-Flash or RiF.

A. Overview of RiF Scheme

The key novelty of the RiF scheme is to employ an in-flash retry engine (which we call the ODEAR engine) which can 1) predict if a sensed page can be unrecoverable by an off-chip ECC decoder and, if so, 2) adjust the read-reference voltages and 3) re-read the same page using the adjusted V_{REF}^{Ri} values. Therefore, the RiF-enabled flash die, incorporating the ODEAR engine, does not waste the flash channel bandwidth for moving and decoding unrecoverable pages. Figs. 8(a) and 8(b) compare the execution flow of read operations when N_{RR} = 1 in a conventional flash die and a RiF-enabled flash die. In the RiF scheme, data transfer and an ECC decoding (2 and 3 in Fig. 8(a)) for an uncorrectable page can be avoided by the on-die ODEAR engine. As shown in Fig. 8(c), therefore, the total execution time of an SSD with RiF-enabled flash dies can be reduced by 126 μs compared to SSD_{one} in Fig. 7(b).

Fig. 9 shows an organizational overview of a RiF-enabled flash die. (For simplicity, only a single plane of a flash die is shown.) The ODEAR engine consists of two main modules: read-retry predictor (RP) and read-voltage selector (RVS). When a read command is issued, a corresponding page is read and stored inside a page buffer. The RP module (which is added to each plane) decides whether the page can be correctable by an off-chip ECC engine (❶ in Fig. 9). If the page is estimated to be correctable, the ready flag of the status register is set to 1 so that the controller knows the read operation is complete (❷). When the page is predicted to be unrecoverable by the off-chip ECC engine, the RP communicates to the RVS so that a read-retry procedure can be initiated on the same page (❸). The RVS module then searches for the optimal voltage level and re-reads the page with the

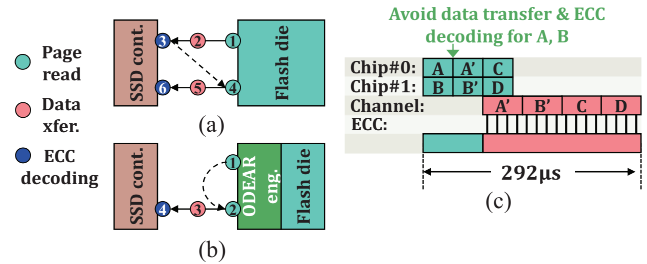


Fig. 8: A comparison of execution flow between (a) a conventional flash die and (b) a RiF-enabled flash die with (c) an example of the execution timeline in a RiF-enabled flash die.

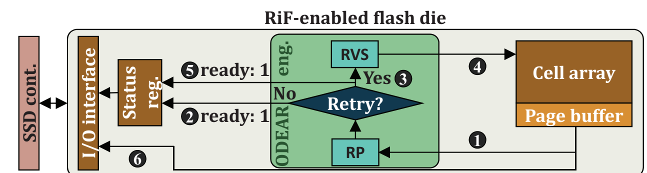


Fig. 9: An organizational overview of a RiF-enabled flash die.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

estimated V_{REF}^{Ri} values (❹). After the re-read operation, the ready flag is set to 1. (❺). The SSD controller then initiates a data transfer operation (❻). Note that the re-read page does not go through the RP module but is directly sent to the off-chip ECC engine.

B. Read-Retry Prediction by RP Module

A straightforward way to implement a RiF-enabled flash die is to integrate an LDPC decoder within the flash die itself. This approach ensures that only error-free data is transferred out after a read-retry procedure is completed within the flash die, thus causing no impact on the channel utilization from a read-retry. However, placing a complex LDPC decoder within a flash die is not practical. As an alternative solution, we place only the RP module, an efficient error-correctability predictor, within the flash die. RP only estimates the RBER of a page that is used to predict the need for a read-retry. As shown in Fig. 3(a), the error correction capability of an LDPC decoder is predetermined. Therefore, if the RBER of a page were to be known in advance, RP could decide whether a read-retry is needed or not *without* a complex LDPC decoding process.

For an efficient read-retry prediction, RP exploits the positive correlation between the RBER and syndrome weight (i.e., $\sum s_k$) of a read page. As explained in Section II-B, each syndrome s_k is obtained by performing XOR operations on a subset of the read data. Let $\mathcal{V}_k$ denote a set of bits used in computing s_k. When no errors are present within $\mathcal{V}_k$, the resulting syndrome s_k is zero. However, as the RBER of the page increases, it is more likely that bits in $\mathcal{V}_k$ experience errors, thus leading to a higher probability of s_k being to one. Consequently, the syndrome weight, which is the sum of all the syndromes, increases with an increasing RBER value. The RP module takes a simple heuristic in predicting if a sensed page can be successfully decoded by an ECC decoder by using the computed syndrome weight of the page. When the syndrome weight of a page is larger than the correctability threshold value ρ_s, RP predicts that the page cannot be successfully decoded by an off-chip ECC engine, thus requiring an invocation of an in-flash read-retry procedure.

The key challenge of designing the RP module, therefore, is how to choose ρ_s. In deciding ρ_s, we exploit a strong correlation between RBER values and syndrome weights. When a sensed page is unrecoverable by an off-chip ECC decoder, its RBER value becomes larger than the error correction capability of the ECC decoder. In the current RP heuristic, we assume that an RBER value of a sensed page and its syndrome weight have a 1:1 relationship, so when we know a syndrome weight, we can estimate its RBER value. Therefore, when the syndrome weight of a sensed page is larger than ρ_s, RP decides that the sensed page is unrecoverable by an off-chip ECC decoder. Fig. 10 shows how strongly an RBER value and its syndrome weight are related in QC-LDPC. Since the correction capability of the QC-LDPC was set to 0.0085 (see Fig. 3), we set ρ_s to the corresponding syndrome weight for the RBER value of 0.0085.

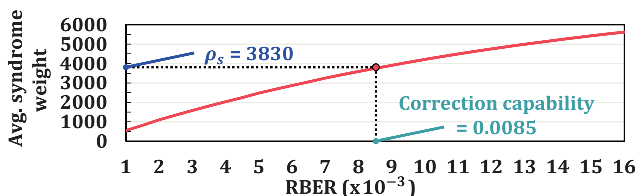


Fig. 10: A correlation analysis between RBER and syndrome weight.

In order to evaluate the adequacy of RP as a read-retry predictor, we performed a validation study of RP for pages with feasible RBER values. In this paper, we assumed that programmed flash blocks are refreshed every month, so we evaluated the accuracy of RP over the range of RBERs for pages with retention times up to one month.[3] For each RBER value, we generated 10^5 test pages with the same RBER value and compared the retry prediction result from RP with the real read-retry result from the QC-LDPC decoder. Fig. 11 shows the summary of our validation results. RP achieves a high prediction accuracy of 99.1% on average when RBER values are above the correction capability, which indicates that RP can correctly detect the majority of uncorrectable pages.

When an RBER value is close to the correction capability of an ECC decoder, however, we noticed that the prediction accuracy of RP gets degraded. For example, when the RBER value of a page is equal to the correction capability, the accuracy of RP drops to 50.3%. Consequently, when the RBER of a page is close to the correction capability, RP makes two types of mispredictions. First, RP may incorrectly identify uncorrectable data as correctable, thereby preventing the complete elimination of uncorrectable data movements. Such uncorrectable data transfers to an off-chip ECC decoder, however, are negligible because the RBER range for poor RP accuracy is less than 2% of the overall RBER range under the 1-month data retention requirement. Second, RP may initiate unnecessary read-retry operations on pages that are actually correctable. The key concern for this case is whether the re-read page with an adjusted V_{REF}^{Ri} is not recoverable by an off-chip ECC decoder. However, the RBER of the page is lowered when a read-retry is performed [23], [32], [46], so it is unlikely that the re-read page is unrecoverable by the ECC decoder.

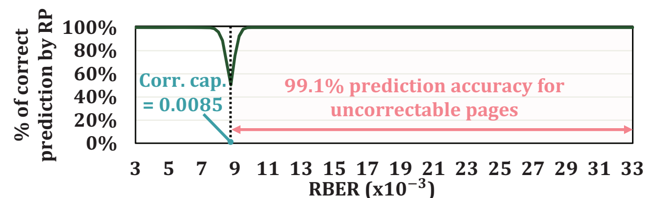


Fig. 11: Validation results of RP over LDPC decoder.

[3] Modern SSDs typically refresh stored data periodically for reliability management [52], [53].

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

C. Read Voltage Selector

Upon predicting that a page is uncorrectable, RP initiates RVS to choose better V_{REF}^{Ri} values for the page. In order to adjust the V_{REF}^{Ri} values, a RiF-enabled flash die incorporates Swift-Read [32], which finds near-optimal V_{REF}^{Ri} values without requiring controller assistance. Unlike the conventional usage of issuing a Swift-Read command after an ECC decoding failure, in RiF, the RVS module internally issues a Swift-Read command for uncorrectable pages. After a Swift-read is issued by RVS, the RiF-enabled flash die finds the near-optimal voltage values for the uncorrectable page and re-reads the page with those values. After the page is re-read, the sensed page does not go through the RP module again. This decision is largely dependent on the efficiency of a Swift-Read command, which claims to locate near optimal V_{REF}^{Ri} values.[4]

V. Implementation of RiF-enabled Flash

Although the calculation of a syndrome weight is less complex than a complete LDPC decoding process, its efficient implementation at the flash die level remains challenging. First, computing a syndrome weight introduces additional read latency. To compute a syndrome weight, data inside a page buffer should be read and processed by RP after completing a page sensing operation. Considering a latency for a page sensing operation takes 40 μs and a latency for reading a 16-KiB page from the page buffer takes 10 μs [43], the read latency increases by at least about 25%. Second, computing a syndrome weight requires complex bitwise operations which are not straightforward to implement within a flash die. As shown in Fig. 2, the bits needed to compute a syndrome are irregularly distributed over an entire codeword. Consequently, a direct implementation of a syndrome computation typically operates in a bitwise serial manner, thus making it quite challenging to efficiently implement in hardware.

To address the above implementation challenges, we employ two optimizations at two levels of RP implementation. First, to minimize the latency of computing a syndrome weight, we reduce the number of syndromes computed (Section V-A). Second, to optimize an efficiency of a syndrome computation, we reorganize a codeword layout in a hardware-friendly fashion (Section V-B).

A. Approximate Syndrome Computation

1) Chunk-based Prediction: The first approximation used in our RP implementation is based on our key observation that the errors within the same page are uniformly (or randomly) distributed. Fig. 12 shows RBER values of chunks of the same size within the same 16-KiB page, where the chunk sizes are 4-KiB, 2-KiB, and 1-KiB.[5] For each chunk size, we compare a ratio between the maximum bit error rate

[4]When the efficiency of a Swift-Read command is not as good as reported in [32], it is straightforward to modify for RP to predict the read-retry of the second sensed page.

[5]Fig. 12 summarizes our characterization results using more than 10^5 pages from 160 TLC NAND flash chips to cope with multiple kinds process variation of flash memory [54] in a statistically meaningful manner, under various operating conditions.

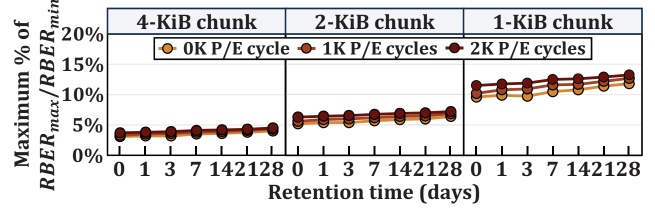


Fig. 12: RBER similarity among fixed-size chunks within the same 16-KiB page.

$RBER_{max}$ and the minimum bit error rate $RBER_{min}$ among chunks. As shown in Fig. 12, RBER similarity exists between chunks for all chunk size configurations. For example, we observed that the maximum difference between $RBER_{max}$ and $RBER_{min}$ for the 4-KiB chunk size is only up to 4.5%, indicating that only marginal error characteristic variations exist among chunks within a page. This is because the data randomization technique [55], [56] of modern flash memory, which is common practice to reduce the probability of worst-case data patterns, randomly distributes the cell's V_{TH} states within a page regardless of the original data values to store. In addition, cells on the same page tend to have similar reliability characteristics due to the strong process similarities [19], [23], [57]. By exploiting a strong intra-page RBER similarity, RP checks only a single chunk corresponding to the size of a single codeword within a page, thus efficiently reducing the number of syndromes computed.

Despite the observed RBER similarity between chunks, the results in Fig. 12 also indicate the importance of sufficient sample size in a chunk in order to achieve practicable prediction accuracy. As the chunk size decreases, the ratio of $RBER_{max}$ to $RBER_{min}$ increases. For example, a relatively high RBER variance was observed for a chunk size of 1-KiB (up to 13.5%). Although a smaller chunk size would reduce RP's overhead even further, we chose a 4-KiB chunk size for RP design to minimize misprediction overhead while retaining sufficient performance benefits.

2) Syndrome Pruning: To further reduce the overhead of computing a syndrome, we employ syndrome pruning, which skips the computation of redundant syndromes. Fig. 13 depicts a parity check matrix of QC-LDPC. In QC-LDPC, the parity check matrix $\boldsymbol{H}$ is a r by c block matrix composed of t by t sub-matrices ($\boldsymbol{Q}(C_{i,j})$) [39], [40], [58]. Each sub-matrix is a circulant matrix that can be obtained by cyclically shifting the identity matrix to the right, where the shifting coefficient is denoted as $C_{i,j}$. For example, $\boldsymbol{Q}(1)$ can be obtained by shifting the identity matrix by 1. As shown, there are r times t syndromes in QC-LDPC. Among them, we utilize only the first

$$\boldsymbol{H} = \underbrace{\begin{bmatrix} \boldsymbol{Q}(C_{1,1}) & \boldsymbol{Q}(C_{1,2}) & \cdots & \boldsymbol{Q}(C_{1,c}) \\ \boldsymbol{Q}(C_{2,1}) & \boldsymbol{Q}(C_{2,2}) & \cdots & \boldsymbol{Q}(C_{2,c}) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{Q}(C_{r,1}) & \boldsymbol{Q}(C_{r,2}) & \cdots & \boldsymbol{Q}(C_{r,c}) \end{bmatrix}}_{N = c \times t} \; (M = r \times t) \qquad \boldsymbol{Q}(1) = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}}_{t} \; (t)$$

Fig. 13: A parity check matrix of QC-LDPC.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

t syndromes (i.e., s_k $(0 \le k \le t-1)$) for prediction purposes.[6] These syndromes are explicitly represented by the first row of a block matrix denoted as $\boldsymbol{H}$ in Fig. 13 (i.e., $Q(C_{1,1})$ to $Q(C_{1,c})$). The rationale for using only the first t syndromes for prediction comes from the fact that the remaining syndromes (i.e., s_k $(t \le k \le rt-1)$) merely serve to reconfigure the bit arrangements of the first t syndromes. By using only the first t syndromes for prediction, the number of syndrome computations can be reduced by a factor of r. Moreover, this approach does not significantly compromise the effectiveness of the prediction, as the remaining syndromes do not provide substantial new information.

Fig. 14 shows the evaluation result of RP when two proposed approximation schemes are employed. Although the accuracy drops slightly, it still maintains a high prediction accuracy of 98.7%.

B. On-Die Syndrome Computation

In order to calculate a syndrome, a set of bits that are represented by a row of parity check matrix $\boldsymbol{H}$ should be XORed, as explained in Section II-B. However, as shown in Fig. 13, $\boldsymbol{H}$ of QC-LDPC consists of identity matrices shifted by coefficients ($C_{i,j}$). Consequently, the bits needed to compute a syndrome are distributed irregularly across different segments constituting the codeword. Therefore, computing the syndrome inside the flash die should involve complex bitwise processing, which increases the area and latency of RP.

To facilitate the on-die syndrome computation, we propose a codeword rearrangement scheme that makes $\boldsymbol{H}$ a hardware-friendly structure. Fig. 15 shows how rearranging the codeword simplifies $\boldsymbol{H}$.[7] Since the row length of $\boldsymbol{H}$ is c times t, the codeword can be divided into c segments. Here, the segment i corresponds to $Q(C_{1,i})$, which is the identity matrix shifted to the right by $C_{1,i}$, so if the segment i is rotated to the left by $C_{1,i}$, then $Q(C_{1,i})$ is *logically* equivalent to the identity matrix. For example, if the segment 1 is rotated to the left by $C_{1,1}$, the corresponding sub-matrix $Q(C_{1,1})$ becomes (logically) equal to $Q(0)$ (i.e., identity matrix). The proposed codeword rearrangement scheme rotates all segments to the left by corresponding shifting coefficients. By doing so, the parity check matrix is simplified to c identity sub-matrices, reducing the syndrome computation to a simple process of counting the number of 1's resulting from the XOR operations executed between segments.

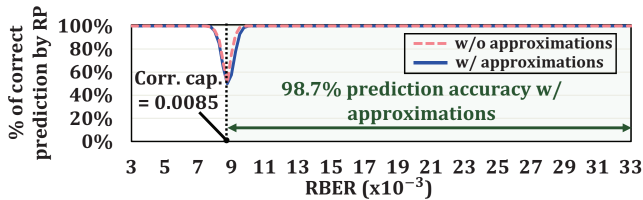


Fig. 14: A comparison of RP prediction accuracy with two approximation techniques.

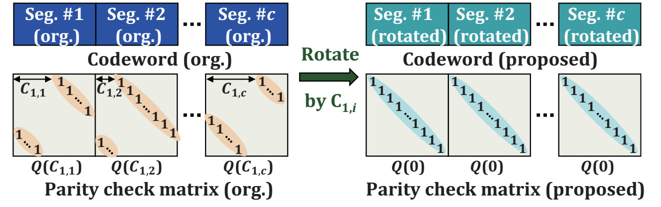


Fig. 15: A simplified parity check matrix when the proposed codeword layout is employed.

The proposed codeword rearrangement scheme is only applied when making predictions inside the flash die while LDPC decoding should be performed on the original codeword layout. To support an off-chip LDPC decoding as well as an on-die syndrome computation, the flash controller changes the layout of the codeword before sending data (i.e., after ECC encoding) to the flash die for writing and restores the layout when it receives data (i.e., before ECC decoding) from the flash die for reading. Note that bitwise rotation is an essential behavior of QC-LDPC, so the LDPC engine is already equipped with several barrel shifters for rotating segments [59]–[61].

Fig. 16 shows an overall organization of RP, assuming the page buffer is organized in 128-bit words [62]. Although syndrome computation is straightforward, a naive implementation of RP may incur significant overhead due to the large segment size (e.g., 1024-bit). For efficient implementation, RP divides the entire syndrome into 128-bit units and accumulates the weight in turn. To calculate the first 128 syndromes, for example, RP fetches the first 128-bit of data from each segment into 128-bit segment_reg (❶ in Fig. 16) while fetched data is sequentially XORed and stored in 128-bit syndrome_reg (❷). After a series of XORs to compute the 128 syndromes, the weight of the syndromes is counted and accumulated in the accumulator (❸). RP repeats the above process until the total syndrome weight is accumulated, and then completes the read-retry prediction by comparing the syndrome weight to ρ_s (❹). Because each stage of the syndrome computation is fully pipelined (i.e., the latency of ❷ and ❸ overlaps with ❶), the time required to retrieve a chunk from the page buffer determines the overall latency of a read-retry prediction. That

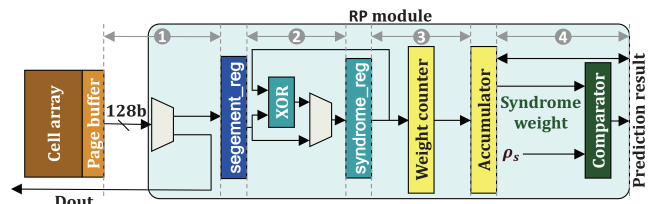


Fig. 16: An overall organization of RP.

[6]The parity check matrix $\boldsymbol{H}$ of QC-LDPC used in this paper is a 4 by 36 block matrix composed of 1024 by 1024 submatrices. Therefore, we compute only 1024 syndromes out of 4096 syndromes.

[7]In Fig. 15, only the first row of the block matrix $\boldsymbol{H}$ is shown. This is because the rest of the syndromes are not computed when the syndrome pruning technique is applied.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

is, Assuming a 4-KiB chunk, it takes about 2.5 μs [43].

VI. Evaluations

A. Experimental Setup

We evaluated the effectiveness of the RiF scheme on improving system I/O performance using MQSim-E [63], a state-of-the-art SSD simulator used for evaluating modern enterprise SSDs. To accurately simulate the error characteristics of modern SSDs, we extended the NAND flash model of MQSim-E to match real-device characterization results described in Section III. Each block in MQSim-E was configured to match the device characterization results of one of the real tested blocks. For example, each block in MQSim-E is modeled with a lookup table that contains RBER values at different P/E-cycle counts, retention ages, and block read counts from the device characterization results of a randomly chosen test block. When a page is read in the extended MQSim-E, the RBER value of the target page is determined from the RBER lookup table of the block (where the page belongs to) based on the current operating parameters of the block (e.g., a P/E cycle count and a data retention time). Based on the estimated RBER value, the extended MQSim-E decides whether the target page requires a read-retry procedure. When a read-retry procedure is required for the page, the estimated RBER value is used in deciding tECC (which varies over the RBER value of the page). To simulate the RP module of a RiF-enabled flash chip, a probability-based model is used using the RP prediction accuracy function (given as a function of an RBER value) as shown in Fig. 14.

Table I summarizes our evaluated SSD configurations. We assumed a 2-TiB SSD with 8 channels, 4 dies per channel, and 4 planes per die. Each plane consisted of 1,888 blocks, and each block had 576 16-KiB pages. tR, tPROG, and tBERS are set to 40 μs, 400 μs, and 3.5 ms, respectively, for our simulated NAND flash chip, based on the real NAND flash chips used in our characterization study[8]. We set the host interface to support a maximum bandwidth of 8.0 GB/s as specified by the PCI Express (PCIe) 4.0 standard [50]. The I/O bandwidth of a flash channel supports 1.2 Gb/s peak bandwidth (i.e., tDMA = 13 μs for a 16-KiB page). With 8 channels, therefore, our simulated SSD fully supports the peak bandwidth of the host interface (i.e., 1.2 GB/s $\times$ 8 $>$ 8.0 GB/s). We assumed a 4-KiB QC-LDPC engine with a correction capability of 0.0085, and tECC varied from 1 μs to 20 μs depending on the target page's RBER. When the target page is successfully decoded following V_{REF} adjustment, we assume tECC to be 1 μs because the RBER value of a sensed page with near-optimal V_{REF} is significantly less than the ECC capability [46]. The latency of the RP module, tPRED, is set to 2.5 μs.

TABLE I: Evaluated SSD configurations.

Configuration	2-TiB total capacity; 8 channels; 4 dies/channel; 4 planes/die; 1888 blocks/plane; 576 pages/block
Latencies (μs)	tR = 40; tPROG = 400; tBERS = 3500; tDMA = 13; tECC = 1 to 20; tPRED = 2.5
Bandwidth	8.0 GB/s external I/O bandwidth (PCIe 4.0, 4-lane); 1.2 GB/s channel I/O bandwidth
ECC engine	4-KiB LDPC with 0.0085 correction capability

[8]The timing parameters tPROG and tBERS define the latency for the program and erase operations, respectively.

We conducted our experiments using eight workloads obtained from two I/O trace sets: AliCloud traces [51] and Systor traces [64]. The AliCloud traces consist of 1,000 block I/O traces collected from a cloud block storage system over one month. On the other hand, the Systor traces comprise 44 block I/O traces gathered from a cloud block storage system over a period of 28 days. From these trace sets, we carefully selected eight representative traces. First, we categorized the I/O traces based on their read ratios. From each category of I/O traces with similar read ratios, we selected the ones with the highest total I/O size. In the case of the Systor traces, we found that all traces had similar read ratios and I/O intensities, so we only included the first two traces (Sys_0, Sys_1).

Table II summarized the key I/O characteristics of eight traces. The read ratio indicates the proportion of read requests among all I/O requests while the cold read ratio represents the fraction of read requests to pages that have not been updated at all during workload simulation. For example, in Ali_{124}, 96% of all I/O requests are read requests. Among the read requests, 79% are read to pages that are not updated at all during the workload simulation. When a read request is a cold read, a read retry is more likely because of its long retention time. In general, the higher the cold read ratio, the higher the frequency of read retries.

To evaluate the effectiveness of the proposed RiF scheme, we built RiFSSD that employs RiF-enabled flash chips. Furthermore, to compare the performance of the RiF scheme with existing state-of-the-art read mitigation techniques, we built four more SSDs, SENC, SWR, SWR+, and RPSSD. SENC and SWR are SSDs based on two state-of-art techniques, Sentinel and Swift-Read (as described in Section III-B), respectively. To provide a more competitive comparison, we have included SWR+ and RPSSD. SWR+ is an SSD that combines SWR with an advanced V_{REF} tracking scheme [19]. By proactively employing pre-optimized V_{REF} values, SWR+ complements the reactive nature of SWR, thereby decreasing the frequency of read-retry procedures. RPSSD is an alternative RiF scheme design that can be implemented if applying architectural modifications to an existing flash chip is challenging. It incorporates an RP module at the SSD controller level to terminate prolonged ECC decoding when a sensed page is predicted to be uncorrectable. To evaluate RiFSSD in the scenario where the existing off-chip read-retry solutions work best, we assume

TABLE II: Key I/O characteristics of eight I/O traces.

Workload	Read ratio	Cold read ratio	Workload	Read ratio	Cold read ratio
Ali_2	0.27	0.50	**Ali_{124}**	0.96	0.79
Ali_{46}	0.34	0.75	**Ali_{295}**	0.42	0.73
Ali_{81}	0.43	0.74	**Sys_0**	0.70	0.82
Ali_{121}	0.92	0.70	**Sys_1**	0.72	0.83

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

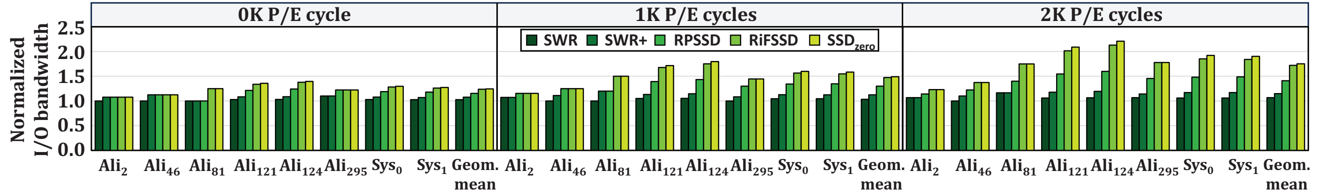


Fig. 17: Comparisons of I/O bandwidth for eight workloads under three different P/E cycles.

that all SSDs can identify the near-optimal V_{REF} value after one read-retry loop.

B. Performance Evaluation

Fig. 17 compares the I/O bandwidth of four SSD configurations, normalized to SENC, under different operating conditions. (Note that SSD_{zero} represents a hypothetical SSD that experiences no read retry.) When compared to state-of-the-art read-retry optimization techniques, RiFSSD outperforms SENC, SWR and SWR+ by significantly improving the average I/O bandwidth at all three P/E cycles. For example, RiFSSD improves the I/O bandwidth at 2K P/E cycles by 72.1%, 61.2% and 50.0% over SENC, SWR, and SWR+, respectively. Furthermore, the average I/O bandwidth of RiFSSD is comparable to that of the ideal SSD_{zero}, with the maximum difference of 1.8% at 2K P/E cycles.

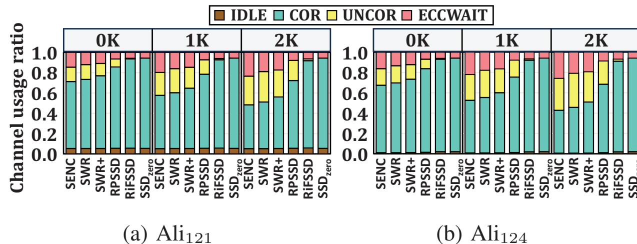
To better understand the efficiency of RiFSSD, we compared how flash channels were used in each SSD using two workloads with the highest read ratio, Ali_{121} and Ali_{124}. As shown in Fig. 18, the usage of a flash channel is categorized into four types: IDLE indicates when the flash channel is not used, COR and UNCOR represent a time interval when correctable and uncorrectable pages are transferred to an off-chip ECC decoder, respectively, and ECCWAIT is a special case of IDLE when the flash channel is idle because of on-going ECC decoding operations. As shown in Fig. 18, SENC, SWR, and SWR+ waste a considerable share of the flash channel bandwidth while 1) transferring uncorrectable pages to an off-chip ECC decoder and 2) taking a long latency to decode uncorrectable pages. For example, in Ali_{124}, 54.4% of the flash channel bandwidth was wasted in UNCOR and ECCWAIT at 2K P/E cycles in SWR because of a large number of cold reads that require read-retry procedures. While RPSSD effectively reduces wasted channel bandwidth from ECCWAIT, it still suffers unnecessary data transfers of uncorrectable pages to the off-chip RP module. On the other hand, RiFSSD consumed nearly all of the channel bandwidth under COR. In Ali_{121}, for example, at 2K P/E cycles (when read-retries are most frequent), RiFSSD and RPSSD wasted 1.8% and 19.9% of the flash channel bandwidth, respectively, under UNCOR.

To understand the benefit of reduced read retries on SSD-level read latencies, we compared five SSD configurations using their SSD-level read latencies. As shown in Fig. 19, the proposed RiF scheme was effective at reducing the read tail latency, a crucial performance factor for many data-intensive applications [65]–[71]. For example, in RiFSSD, the 99.99-th percentile tail latency in Ali_{124} was reduced by 91.8%, 82.6%, and 56.3% at 2K P/E cycles over SENC, SWR, and SWR+, respectively.

C. Overhead Evaluation

We estimated the area/power overhead of the RP module from a synthesis result using Synopsys Design Compiler. Although RiF-enabled flash dies necessitate the use of both the RVS and RP modules, one of our baseline SSDs, SWR, already employs hardware logic such as the RVS module to enable Swift-Read commands at the flash die level. Therefore, in this evaluation, we focus on the overhead of the RP module.

In the 130 nm process at a 100 MHz operating frequency, the total area and power consumption of the RP module were only 0.012 mm^2 and 1.28 mW, respectively. Compared to the typical area of modern flash dies (e.g., 101 mm^2 in [72]), the space overhead of the RP module seems to be negligible. Although the RP module consumes slightly more power/energy when reading a correctable page because of the additional RP module, the RiF scheme is effective in minimizing the overall energy consumption of a modern SSD with frequent read retries. For example, when a read retry procedure is needed, the RiF scheme reduces the energy consumption of a read request by about 907 nJ [73] (which is needed for an unrecoverable page transfer) while the RP



(a) Ali_{121} (b) Ali_{124}

Fig. 18: Channel usage breakdown results in Ali_{121} and Ali_{124}.

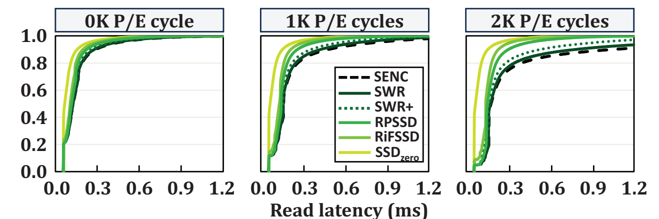


Fig. 19: Cumulative distributions of read latencies in Ali_{124}.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

module can increase the energy consumption by about 3.2 nJ only for its read-retry prediction.

VII. Related Work

To our knowledge, this work is the first to predict a read failure *inside* NAND flash memory, which significantly mitigates the performance overhead due to read-retry in modern SSDs. We provide a brief overview of closely related work that aims to 1) reduce the read-retry overhead and 2) leverage the computation capability of NAND flash chips (i.e., *in-flash processing*) to improve the performance of storage systems.

Read-Retry Optimization. Due to the significant performance impact of read-retry in modern SSDs, a large body of prior work has attempted to optimize the read-retry procedure [20], [21], [23], [27]–[29], [31], [32], [46], [74]–[77]. Many existing read-retry optimizations aim to reduce N_{RR} by accurately determining near-optimal V_{REF} values based on 1) sophisticated V_{TH} models of NAND flash memory [20], [31], [74]–[76] or 2) pre-optimized V_{REF} values that have enabled successful decoding of previously-read pages [19], [27]–[29]. However, their effectiveness is limited in 3D TLC and QLC NAND flash memory due to the rapid V_{TH} shift and complex error characteristics of modern NAND flash memory.

More recent studies have proposed to use the error patterns of certain (sampled) flash cells of the target page for accurate prediction of near-optimal V_{REF} values [23], [32], [77]. While these techniques successfully reduce N_{RR} to around one iteration (e.g., 1.2 iteration on average [23]), they cannot be applied proactively to all reads due to the additional page read latency introduced by reading sample data. In contrast, RiF integrates proactive error prediction and V_{REF} adjustment at the flash-die level, effectively minimizing the read-retry overhead without introducing significant additional latency.

In-Flash Processing. Several works [62], [78]–[80] have demonstrated the effectiveness of in-flash processing at mitigating internal data-movement bottleneck in modern SSDs. Existing in-flash processing techniques offload simple (lightweight) computations to NAND flash dies, which significantly reduces off-chip data movement by transferring only the computation results from NAND flash dies. Flash-Cosmos [80] and ParaBit [78] enable a NAND flash die to perform bulk bitwise operations internally and transfer only the results to the flash controller through the flash channel. PiF [79] implements a computation unit inside a NAND flash die for simple filtering tasks (e.g., a pattern matching function), which avoids transferring unnecessary data in target pages to the flash controller, thereby significantly reducing SSD-internal data movement. While existing techniques primarily focus on enhancing the performance of specific applications, to our knowledge, RiF is the first to leverage in-flash processing to improve I/O performance in general storage systems.

VIII. Conclusions

We have presented the RiF scheme, a novel read-retry solution that minimizes unnecessary data transfers between a flash chip and an off-chip ECC decoder during read-retry procedures, thus significantly reducing the overhead of read-retries in modern high-performance SSDs. The key observation behind the proposed RiF scheme is that whether a read retry is needed or not can be accurately predicted without decoding a sensed page by an off-chip ECC decoder. We proposed an accurate but efficient syndrome weight-based read-retry predictor which can be implemented inside a flash die as a part of the ODEAR engine that supports an efficient read voltage selector as well. Our experimental results using a RiF-aware SSD, RiFSSD, show that RiFSSD can improve the effective SSD I/O bandwidth by 72.1% on average over existing state-of-art read-retry solutions at 2K P/E cycles. Furthermore, thanks to a highly-optimized implementation of the ODEAR engine, a RiF-aware flash chip incurs negligible power and area overheads over existing flash chips.

Acknowledgments

We thank the anonymous reviewers of HPCA 2024 for their valuable comments that greatly improved our paper. This work was supported by MOTIE (Ministry of Trade, Industry & Energy) (1415181081) and KSRC (Korea Semiconductor Research Consortium) (20019402). Jisung Park was supported by the National Research Foundation of Korea (RS-2023-00283799). The ICT at Seoul National University provided research facilities for this study. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. (Corresponding author: Jihong Kim.)

References

[1] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E. Priebe, and Alexander S. Szalay. FlashGraph: processing billion-node graphs on an array of commodity SSDs. In *FAST*, 2015.
[2] Hang Liu and H. Howie Huang. Graphene: Fine-grained IO Management for Graph Computing. In *FAST*, 2017.
[3] Nima Elyasi, Changho Choi, and Anand Sivasubramaniam. Large-Scale Graph Processing on Emerging Storage Devices. In *FAST*, 2019.
[4] Kiran K. Matam, Gunjae Koo, Haipeng Zha, Hung-Wei Tseng, and Murali Annavaram. GraphSSD: Graph Semantics Aware SSD. In *ISCA*, 2019.
[5] Hyeokjun Choe, Seil Lee, Hyunha Nam, Seongsik Park, Seijoon Kim, Eui-Young Chung, and Sungroh Yoon. Near-data Processing for Machine Learning. *arXiv*, 2016.
[6] Shengwen Liang, Ying Wang, Youyou Lu, Zhe Yang, Huawei Li, and Xiaowei Li. Cognitive SSD: A Deep Learning Engine for In-Storage Data Retrieval. In *USENIX ATC*, 2016.
[7] Sasmung. PM9A3, 2023. https://www.samsung.com/semiconductor/global.semi.static/[S210809]_PM9A3_SSD_Whitepaper.pdf.
[8] NETINT. Codensity D400, 2019. https://www.netint.cn/wp-content/uploads/2019/08/NETINT_Codensity_D400_SSD_Product_Brief_19PB003.pdf.
[9] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proceedings of the IEEE*, 2017.
[10] Micron. Micron 3D NAND Flash Memory, 2016.
[11] Jiangli Zhu. High-Throughput LDPC Solution for Reliable and High Performance SSD. In *Flash Memory Summit*, 2016.
[12] Shiuan-Hao Kuo. Novel 4K Error Correcting Code for QLC NAND. In *Flash Memory Summit*, 2017.
[13] Shiuan-Hao Kuo. Ultra MMI : an LDPC decoder that doubles throughput at end-of-life. In *Flash Memory Summit*, 2019.
[14] Jiho Kim, Myoungsoo Jung, and John Kim. Decoupled SSD: Reducing Data Movement on NAND-Based Flash SSD. *IEEE CAL*, 2021.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

[15] Narges Shahidi, Mahmut T. Kandemir, Mohammad Arjomand, Chita R. Das, Myoungsoo Jung, and Anand Sivasubramaniam. Exploring the Potentials of Parallel Garbage Collection in SSDs for Enterprise Storage Systems. In *SC*, 2016.

[16] Fei Wu, Jiaona Zhou, Shunzhuo Wang, Yajuan Du, Chengmo Yang, and Changsheng Xie. FastGC: Accelerate Garbage Collection via an Efficient Copyback-Based Data Migration in SSDs. In *DAC*, 2018.

[17] Duwon Hong, Myungsuk Kim, Jisung Park, Myoungsoo Jung, and Jihong Kim. Improving SSD Performance Using Adaptive Restricted-Copyback Operations. In *NVMSA*, 2019.

[18] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery. *arXiv*, 2017.

[19] Youngseop Shim, Myungsuk Kim, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim. Exploiting Process Similarity of 3D Flash Memory for High Performance SSDs. In *MICRO*, 2019.

[20] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-recovery and Temperature Awareness. In *HPCA*, 2018.

[21] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation. In *SIGMETRICS*, 2018.

[22] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Optimizing Read-Retry Latency of NAND Flash Memory by Fully Exploiting Advanced Features of Modern SSDs. *arXiv*, 2020.

[23] Qiao Li, Min Ye, Yufei Cui, Liang Shi, Xiaoqiang Li, Tei-Wei Kuo, and Chun Jason Xue. Shaving Retries with Sentinels for Fast Read over High-Density 3D Flash. In *MICRO*, 2020.

[24] Nikolaos Papandreou, Haralampos Pozidis, Thomas Parnell, Nikolas Ioannou, Roman Pletka, Sasa Tomic, Patrick Breen, Gary Tressler, Aaron Fry, and Timothy Fisher. Characterization and Analysis of Bit Errors in 3D TLC NAND Flash Memory. In *IRPS*, 2019.

[25] Qiao Li, Liang Shi, Yufei Cui, and Chun Jason Xue. Exploiting Asymmetric Errors for LDPC Decoding Optimization on 3D NAND Flash Memory. *IEEE TC*, 2019.

[26] Yachen Kong, Meng Zhang, Xuepeng Zhan, Rui Cao, and Jiezhi Chen. Retention Correlated Read Disturb Errors in 3-D Charge Trap NAND Flash Memory: Observations, Analysis, and Solutions. *IEEE TCAD*, 2020.

[27] Borja Peleato, Rajiv Agarwal, John Cioffi, Minghai Qin, and Paul H Siegel. Towards Minimizing Read Time for NAND Flash. In *GLOBECOM*, 2012.

[28] Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Thomas Mittelholzer, Evangelos Eleftheriou, Charles Camp, Thomas Griffin, Gary Tressler, and Andrew Walls. Using Adaptive Read Voltage Thresholds to Enhance the Reliability of MLC NAND Flash Memory Systems. In *GLSVLSI*, 2014.

[29] Yu Cai, Yixin Luo, Erich F Haratsch, Ken Mai, and Onur Mutlu. Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery. In *HPCA*, 2015.

[30] Meng Zhang, Fei Wu, Qin Yu, Weihua Liu, Yifan Wang, and Changsheng Xie. Exploiting Error Characteristic to Optimize Read Voltage for 3-D NAND Flash Memory. *IEEE TED*, 2020.

[31] Nikolaos Papandreou, Nikolas Loannou, Thomas Parnell, Roman Pletka, Milos Stanisavljevic, Radu Stoica, Sasa Tomic, and Haralampos Pozidis. Reliability of 3D NAND Flash Memory with a Focus on Read Voltage Calibration from a System Aspect. In *NVMTS*, 2020.

[32] Wanik Cho, Jongseok Jung, Jongwoo Kim, Junghoon Ham, Sangkyu Lee, Yujong Noh, Dauni Kim, Wanseob Lee, Kayoung Cho, Kwanho Kim, Heejoo Lee, Sooyeol Chai, Eunwoo Jo, Hanna Cho, Jong-Seok Kim, Chankeun Kwon, Cheolioona Park, Hveonsu Nam, Haeun Won, Taeho Kim, Kyeonghwan Park, Sanghoon Oh, Jinhyun Ban, Junyoung Park, Jaehyeon Shin, Taisik Shin, Junseo Jang, Jiseong Mun, Jehyun Choi, Hyunseung Choi, Suna-Wook Choi, Wonsun Park, Dongkvu Yoon, Minsu Kim, Junvoun Lim, Chiwook An, Hyunyoung Shirr, Haesoon Oh, Haechan Park, Sungbo Shim, Hwang Huh, Honasok Choi, Seungpil Lee, Jaesuna Sim, Kichana Gwon, Jumsoo Kim, Woopyo Jeong, Jungdal Choi, and Kyo-Won Jin. A 1-Tb, 4b/Cell, 176-Stacked-WL 3D-NAND Flash Memory with Improved Read Latency and a 14.8 Gb/mm2 Density. In *ISSCC*, 2022.

[33] R. Micheloni, A. Marelli, and K. Eshghi. *Inside Solid State Drives (SSDs)*. Springer, 2013.

[34] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *HPCA*, 2017.

[35] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *ICCD*, 2013.

[36] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN*, 2015.

[37] Jaeyong Lee, Myungsunk Kim, Wonil Choi, Sanggu Lee, and Jihong Kim. TailCut: Improving Performance and Lifetime of SSDs Using Pattern-Aware State Encoding. In *DAC*, 2022.

[38] Kyoji Mizoguchi, Shohei Kotaki, Yoshiaki Deguchi, and Ken Takeuchi. Lateral Charge Migration Suppression of 3D-NAND Flash By Vth Nearing for Near Data Computing. In *IEDM*, 2017.

[39] Osso Vahabzadeh. ECC for NAND Flash. In *Flash Memory Summit*, 2017.

[40] Ned Varnica. LDPC Decoding: VLSI Architectures and Implementations (Module 1). In *Flash Memory Summit*, 2013.

[41] Yuluen Wang. Ultra High Throughput LDPC Schemes for SSD. In *Flash Memory Summit*, 2016.

[42] JEDEC. JESD312, 2022. https://www.jedec.org/document_search?search_api_views_fulltext=jesd312.

[43] Chulbum Kim, Doo-Hyun Kim, Woopyo Jeong, Hyun-Jin Kim, Il Han Park, Hyun-Wook Park, JongHoon Lee, JiYoon Park, Yang-Lo Ahn, Ji Young Lee, Seung-Bum Kim, Hyunjun Yoon, Jae Doeg Yu, Nayoung Choi, NaHyun Kim, Hwajun Jang, JongHoon Park, Seunghwan Song, YongHa Park, Jinbae Bang, Sanggi Hong, Youngdon Choi, Moo-Sung Kim, Hyunggon Kim, Pansuk Kwak, Jeong-Don Ihm, Dae Seok Byeon, Jin-Yub Lee, Ki-Tae Park, and Kye-Hyun Kyung. A 512-Gb 3-b/Cell 64-Stacked WL 3-D-NAND Flash Memory. *IEEE JSSC*, 53(1):124–133, 2018.

[44] Yingge Li, Guojun Han, Sanwei Huang, Chang Liu, Meng Zhang, and Fei Wu. Exploiting Metadata to Estimate Read Reference Voltage for 3-D NAND Flash Memory. *IEEE TCE*, 2023.

[45] Jiho Cho, D. Chris Kang, Jongyeol Park, Sang-Wan Nam, Jung-Ho Song, Bong-Kil Jung, Jaedoeg Lyu, Hogil Lee, Won-Tae Kim, Hongsoo Jeon, Sunghoon Kim, In-Mo Kim, Jae-Ick Son, Kyoungtae Kang, Sang-Won Shim, JongChul Park, Eungsuk Lee, Kyung-Min Kang, Sang-Won Park, Jaeyun Lee, Seung Hyun Moon, Pansuk Kwak, ByungHoon Jeong, Cheon An Lee, Kisung Kim, Junyoung Ko, Tae-Hong Kwon, Junha Lee, Yohan Lee, Chaehoon Kim, Myeong-Woo Lee, Jeong-yun Yun, HoJun Lee, Yonghyuk Choi, Sanggi Hong, JongHoon Park, Yoonsung Shin, Hojoon Kim, Hansol Kim, Chiweon Yoon, Dae Seok Byeon, Seungjae Lee, Jin-Yub Lee, and Jaihyuk Song. 30.3 A 512Gb 3b/Cell 7 th-Generation 3D-NAND Flash Memory with 184MB/s Write Throughput and 2.0 Gb/s Interface. In *ISSCC*, 2021.

[46] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Reducing Solid-State Drive Read Latency by Optimizing Read-Retry. In *ASPLOS*, 2021.

[47] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. Reliability Issues in Flash-Memory-Based Solid-State Drives: Experimental Analysis, Mitigation, Recovery. *Inside Solid State Drives (SSDs)*, 2018.

[48] Jaewon Cha and Sungho Kang. Data Randomization Scheme for Endurance Enhancement and Interference Mitigation of Multilevel Flash Memory Devices. *Etri Journal*, 2013.

[49] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *FAST*, 2018.

[50] PCI-SIG. PCI Express M.2 Specification Revision 4.0, Version 1.1, 2022. https://pcisig.com/specifications.

[51] Jinhong Li, Qiuping Wang, Patrick P. C. Lee, and Chao Shi. An In-Depth Analysis of Cloud Block Storage Workloads in Large-Scale Production. In *IISWC*, 2020.

[52] Yejia Di, Liang Shi, Congming Gao, Qiao Li, Chun Jason Xue, and Kaijie Wu. Minimizing Retention Induced Refresh Through Exploiting Process Variation of Flash Memory. *IEEE TC*, 2019.

[53] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman S. Unsal, and Ken Mai. Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime. In *ICCD*, 2012.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.

[54] Yuqian Pan, Haichun Zhang, Mingyang Gong, and Zhenglin Liu. Process-variation effects on 3d tlc flash reliability: Characterization and mitigation scheme. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020.

[55] Michele Favalli, Cristian Zambelli, Alessia Marelli, Rino Micheloni, and Piero Olivo. A scalable bidimensional randomization scheme for tlc 3d nand flash memories. *Micromachines*, 2021.

[56] Chulbum Kim, Jinho Ryu, Taesung Lee, Hyunggon Kim, Jaewoo Lim, Jaeyong Jeong, Seonghwan Seo, Hongsoo Jeon, Bokeun Kim, Inyoul Lee, Dooseop Lee, Pansuk Kwak, Seongsoon Cho, Yongsik Yim, Changhyun Cho, Woopyo Jeong, Kwangil Park, Jin-Man Han, Duheon Song, Kyehyun Kyung, Young-Ho Lim, and Young-Hyun Jun. A 21 nm High Performance 64 Gb MLC NAND Flash Memory With 400 MB/s Asynchronous Toggle DDR Interface. *IEEE JSSC*, 2012.

[57] Jui-Nan Yen, Yao-Ching Hsieh, Cheng-Yu Chen, Tseng-Yi Chen, Chia-Lin Yang, Hsiang-Yun Cheng, and Yixin Luo. Efficient Bad Block Management with Cluster Similarity. In *HPCA*, 2022.

[58] Youngjoo Lee, Jaehwan Jung, and Incheol Park. Energy-Scalable 4KB LDPC Decoding Architecture for NAND-Flash-Based Storage Systems. *IEICE Transactions on Electrons*, 2016.

[59] Kiran Gunnam. LDPC Decoding: VLSI Architectures and Implementations (Module 2). In *Flash Memory Summit*, 2013.

[60] Li-Wei Liu, Yen-Chin Liao, and Hsie-Chia Chang. Up-gdbf: A 19.3 gbps error floor free 4kb ldpc decoder for nand flash applications. *IEEE Open Journal of Circuits and Systems*, 2022.

[61] Thien T. Nguyen-Ly, Tushar Gupta, Manuel Pezzin, Valentin Savin, David Declercq, and Sorin Cotofana. Flexible, cost-efficient, high-throughput architecture for layered ldpc decoders with fully-parallel processing units. In *2016 Euromicro Conference on Digital System Design (DSD)*, 2016.

[62] Han-Wen Hu, Wei-Chen Wang, Yuan-Hao Chang, Yung-Chun Lee, Bo-Rong Lin, Huai-Mu Wang, Yen-Po Lin, Yu-Ming Huang, Chong-Ying Lee, Tzu-Hsiang Su, et al. ICE: An Intelligent Cognition Engine with 3D NAND-based In-Memory Computing for Vector Similarity Search Acceleration. In *MICRO*, 2022.

[63] Dusol Lee, Duwon Hong, Wonil Choi, and Jihong Kim. MQSim-E: An Enterprise SSD Simulator. *IEEE CAL*, 2022.

[64] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. Understanding Storage Traffic Characteristics on Enterprise Virtual Desktop Infrastructure. In *SYSTOR*, 2017.

[65] Zhang, Jie and Kwon, Miryeong and Gouk, Donghyun and Koh, Sungjoon and Lee, Changlim and Alian, Mohammad and Chun, Myoungjun and Kandemir, Mahmut Taylan and Kim, Nam Sung and Kim, Jihong and Jung, Myoungsoo. FlashShare: Punching Through Server Storage Stack from Kernel to Firmware for Ultra-Low Latency SSDs. In *OSDI*, 2018.

[66] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *FAST*, 2017.

[67] Wonkyung Kang and Sungjoo Yoo. Q-Value Prediction for Reinforcement Learning Assisted Garbage Collection to Reduce Long Tail Latency in SSD. *IEEE TCAD*, 2020.

[68] Timothy Zhu, Michael A Kozuch, and Mor Harchol-Balter. Workload-compactor: Reducing Datacenter Cost While Providing Tail Latency SLO Guarantees. In *SoCC*, 2017.

[69] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. RAIL: Predictable, Low Tail Latency for NVMe Flash. *ACM TOS*, 2022.

[70] Zhibing Sha, Jun Li, Lihao Song, Jiewen Tang, Min Huang, Zhigang Cai, Lianju Qian, Jianwei Liao, and Zhiming Liu. Low I/O Intensity-Aware Partial GC Scheduling to Reduce Long-Tail Latency in SSDs. *ACM TACO*, 2021.

[71] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Reflex: Remote flash $\approx$ local flash. In *ASPLOS*, 2017.

[72] Dongku Kang, Minsu Kim, Su Chang Jeon, Wontaeck Jung, Jooyong Park, Gyosoo Choo, Dong-kyo Shim, Anil Kavala, Seung-Bum Kim, Kyung-Min Kang, et al. 13.4 A 512Gb 3-bit/Cell 3D 6 th-Generation V-NAND flash memory with 82MB/s write throughput and 1.2 Gb/s interface. In *ISSCC*, 2019.

[73] Hyun-Jin Kim, Youngdon Choi, Jangwoo Lee, Jindo Byun, Seungwoo Yu, Daehoon Na, Jungjune Park, Kwangwon Kim, Anil Kavala, Youngmin Jo, Changbum Kim, Sunghoon Kim, Nahyun Kim, Jaehwan Kim, Bongkil Jung, Yena Lee, Chanjin Park, Hansung Joo, Kisung Kim, Yunhee Choi, Pansuk Kwak, Hyeonggon Kim, Jeong-Don Ihm, Dae-Seok Byeon, Jin-Yub Lee, Ki-Tae Park, and Kye-Hyun Kyung. A 1.2V 1.33Gb/s/pin 8Tb NAND Flash Memory Multi-Chip Package Employing F-chip for Low Power and High Performance Storage Applications. In *VLSI*, 2017.

[74] Kin-Chu Ho, Po-Chao Fang, Hsiang-Pang Li, Cheng-Yuan Michael Wang, and Hsie-Chia Chang. A 45nm 6b/cell Charge-Trapping Flash Memory Using LDPC-Based ECC and Drift-Immune Soft-Sensing Engine. In *ISSCC*, 2013.

[75] Frederic Sala, Ryan Gabrys, and Lara Dolecek. Dynamic Threshold Schemes for Multi-Level Non-Volatile Memories. *IEEE TC*, 2013.

[76] Zhengqin Fan, Guofa Cai, Guojun Han, Wenjie Liu, and Yi Fang. Cell-State-Distribution-Assisted Threshold Voltage Detector for NAND Flash Memory. *IEEE COMML*, 2019.

[77] Qiao Li, Min Ye, Yufei Cui, Liang Shi, Xiaoqiang Li, and Chun Jason Xue. Sentinel Cells Enabled Fast Read for NAND Flash. In *HotStorage*, 2019.

[78] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, Jun Yang, and Jiwu Shu. Parabit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs. In *MICRO*, 2021.

[79] Myoungjun Chun, Jaeyong Lee, Sanggu Lee, Myungsuk Kim, and Jihong Kim. PiF: In-Flash Acceleration for Data-Intensive Applications. In *HotStorage*, 2022.

[80] Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu. Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory. In *MICRO*, 2022.

Authorized licensed use limited to: Seoul National University. Downloaded on April 29,2024 at 07:34:22 UTC from IEEE Xplore. Restrictions apply.