Leveraging Partial-Refresh for Performance and Lifetime Improvement of 3D NAND Flash Memory in Cyber-Physical Systems

Jinhua Cui[a,*], Youtao Zhang[b], Liang Shi[c], Chun Jason Xue[d], Jun Yang[b], Weiguang Liu[a], Laurence T. Yang[a,e]

[a] *Huazhong University of Science and Technology*
[b] *University of Pittsburgh*
[c] *East China Normal University*
[d] *City University of Hong Kong*
[e] *St. Francis Xavier University*

Abstract

Three-dimensional (3D) NAND flash memory-based solid state drives (SSDs) have been widely adopted in cyber-physical systems, due to its performance benefits and scalability. Although 3D flash rapidly increases storage capacity by stacking flash cells in the vertical direction, it faces severe retention errors as well. Periodic refreshing, while effectively mitigating the retention issues, seriously degrades the storage performance and the endurance of 3D NAND flash memory.

To address the above challenge, we propose partial-refresh (PR), a novel lightweight data refresh scheme for 3D NAND flash memory in cyber-physical systems. PR leverages LDPC detectability to identify cells that are more vulnerable to errors. By moving these susceptible bits to new pages, we avoid copying an entire page, reduce the refresh cost, and prolong the SSD lifetime. Our experimental results show that, on average, a PR-aware flash memory improves refresh performance by 28.2% and extends the SSD lifetime by 4.6% over the state-of-the-art while preserving the high data reliability.

Keywords: 3D flash memory, retention error, refresh, LDPC, write

*Corresponding author
Email address: cjhnicole@gmail.com (Jinhua Cui)

amplification

2010 MSC: 68-02, 68M01

1. Introduction

In the cyber-physical systems (CPS), massive data processing sets higher demands on the I/O performance and reliability of modern applications [1, 2, 3, 4]. The three-dimensional (3D) NAND flash memory based solid state drives (SSDs) have gained tremendous popularity in CPS, since 3D flash offers higher speed performance and withstands greater shock than traditional hard disk drives (HDDs). However, due to the multidimensional pathes of charge leakage through vertical oxides and lateral spacers in 3D flash, charge loss is much more severe than that in planar flash, resulting in *early retention loss*, a phenomenon that was recently observed by Luo *et al.* [5]. That is, the number of retention errors arises dramatically soon after page programming and then stabilizes for a relative long duration. Data retention error has emerged as one of the major obstacles in manufacturing high storage capacity and high reliability 3D flash memory [6].

To meet the high reliability demand, flash pages widely adopt error correction code (ECC), e.g., low-density parity-check (LDPC), to correct up to k corrupted bits per page. Adopting a stronger ECC increases the error detection and correction capability and thus prolongs the SSD lifetime. However, increasing parity bits introduces higher implementation overhead while achieving diminishing lifetime improvement [7, 8].

A simple yet effective approach for mitigating retention errors is to refresh flash pages to retain data integrity, i.e., it may periodically read the data out, correct possible errors, and then remap the stored data or reprogram the cell in-place before the retention induced errors exceed the ECC error correction capability [7, 8, 9]. However, refresh faces several drawbacks: (1) The refresh requests often originate from the flash translation layer (FTL) so that they may block the normal I/O requests and prevent the SSD from achieving high

performance guarantee for modern applications [7]. (2) The refresh-induced writes consume more program/erase (P/E) cycles of NAND flash memory, and thus degrade the corresponding lifetime of the NAND flash based SSDs.

The existing studies on the flash refresh method can be categorized into two types. One type is to minimize the refresh frequency. Seif *et al.* [10] reduced refresh frequency by accommodating the temperature impact. Luo *et al.* [11] separated the hot and cold data so that the refresh frequency of hot data can be reduced. Di *et al.* [12] minimized the refresh frequency by prioritizing the allocation of high endurance blocks to the data with long retention demand. Liu *et al.* [9] relaxed the retention of data with shorter retention requirements, and thus reduced the refresh frequency. These studies improved the SSD lifetime, but the refresh operation at each refresh interval remains costly. The other type is to optimize the refresh process. Cai *et al.* [8] proposed in-place reprogramming of cells that have more charge to recover the retention-induced leaked electrons. However, recharging a 3D flash page is problematic as it tends to disturb more neighboring pages.

To address the above challenge, we propose partial-refresh (PR), a novel lightweight data refresh scheme for 3D flash memory in dependable cyber-physical systems, that addresses the deteriorated data retention challenge by leveraging LDPC detectability to identify susceptible cells that are more vulnerable to errors. By only moving these susceptible bits to new pages, a PR-aware FTL avoids copying the whole page, reduces the refresh cost and prolongs the flash memory lifetime. Our experimental results show that, on average, the PR-aware FTL improves refresh performance by 28.2% and extends the SSD lifetime by 4.6% over the state-of-the-art while preserving the high data reliability.

In the remaining of this refresh optimization paper, Section 2 reviews the background of refresh in 3D flash and the motivation of this work, and then Section 3 elaborates our proposed partial-refresh approach for 3D NAND flash memory based storage system. Section 4 lists the experimental settings and analyzes the results afterwards. We summarize the related work follow in Section 5. We next present limitations in Section 6. At last, we conclude this article in

Section 7.

2. Background and Motivation

2.1. Refresh in 3D Flash Memory

Instead of shrinking flash cell size continuously in conventional planar flash memory, 3D NAND flash memory achieves density improvement and lithography cost reduction through die stacking [6]. However, data retention errors have emerged as the severe reliability issue for high efficiency 3D NAND flash memory. Due to the multidimensional pathes of charge leakage through vertical oxides and lateral spacers in 3D flash, charge loss is much more serious than that of planar flash memory. One simple and efficient way to guarantee long-term data integrity is to frequently trigger the data refresh method, which periodically reads data out, corrects, and then remaps or in-place reprograms the stored data to avoid data lost [7, 8, 9]. Although guaranteeing flash data integrity, the data refresh method seriously degrades the I/O performance and the endurance of flash based storage system, especially in 3D flash memory. The deteriorated refresh cost comes from two sources:

1) Longer refresh latency at each refresh interval. Since 3D flash memory achieves density improvement through die stacking in the vertical direction, the page count in 3D flash block is greater than planar flash block. This phenomenon, referred as the "*big block*" problem [13, 14], is widely observed in 3D flash memory. For example, Intel presented the 3D flash chip with 6000+ pages per block [15], while planar flash chips usually have 64 or 256 pages per block. We evaluated flash block size effect on the refresh performance with the planar and 3D flash. Fig. 1 presents the comparison of the refresh latency among the planar and 3D flash under different block sizes. The first column of each trace represents a planar flash memory with 256KB block size, and other two columns represent 3D flash chips with different block sizes, i.e. 16MB and 64MB, respectively. As shown in Fig. 1, the refresh cost is proportional to the flash block size. This is because the overhead to perform the refresh method mainly comes

Table 1: Refresh frequency under different life stages in 3D flash memory chip

P/E cycles (K)	refresh frequency
0-1	year
1-2	month
2-4	week

from live page copying. And in 3D flash memory the size of one block increases largely, therefore 3D flash blocks are more likely to contain more valid data, which takes more time to perform the time-consuming refresh operation.

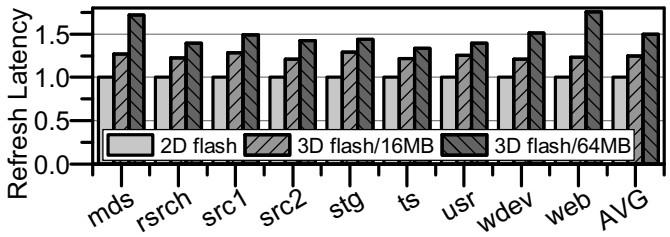

Figure 1: The refresh latency comparison among planar and 3D flash under different block sizes.

2) Increased refresh frequency. Due to the multidimensional pathes of charge leakage through vertical oxides and lateral spacers in 3D flash, charge loss is much more serious than that of planar flash memory. Therefore, the refresh method in 3D flash memory is triggered more frequently. Such frequent refresh will be the dominant source of 3D flash memory write amplification (WA). The increased refresh frequency leads to the deteriorated refresh cost in 3D flash memory. Table 1 collected from [16], shows the supported retention time of 3D NAND flash memory chip in different stages. When the 3D flash memory chip has been erased with 4K P/E cycles, the retention time will decrease to days. Therefore, the frequent and time-consumed refresh is the serious challenge in 3D NAND flash-based storage system.

While preceding discussion shows that refresh seriously degrades the I/O performance and the endurance of 3D flash memory, a limitation of the existing

work focus on addressing this urgent 3D refresh issue. To reduce the refresh overhead, redesigning a novel lightweight data refresh scheme for 3D flash is a better choice.

2.2. Motivation

Fig. 2 illustrates the threshold voltage distribution of two-bit-per-cell MLC flash, where the different amount of charge is used to differentiate the four states, i.e., ER (or erased, 11), P1 (01), P2 (00), and P3 (10) state, respectively. Each state can further divide into two regions, including the *trusted region* (TR) and the *dominating overlap region* (DOR). DOR is sensitive to different types of circuit-level noises, i.e. program disturb noise [17], read disturb noise [18], retention noise [18], and P/E cycling noise [19]. The data in DOR maybe shift to the adjacent state, resulting in unintentionally changing the data stored in one page. These error-prone cells in DOR are also referred to as susceptible cells, while cells in TR are called as non-susceptible cells.

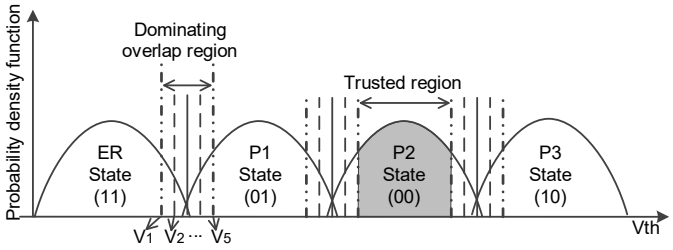

Figure 2: Threshold voltage distribution of MLC flash.

When refresh is invoked at each refresh interval, each valid page content is read out individually. All the bits of a valid page will be read into the flash controller, and error bits will be corrected by the ECC engine, and then all the bits will be redirected into a free flash page. As shown in Fig. 2, the probability of a non-susceptible cell occurring is far beyond a susceptible cell. Assume the data page size of 3D MLC flash is 16KB, and the total number of new error bits appears to be 394 after 1-week retention loss, while the number of the new error bits will be 300 after 2-week retention loss [16]. When refresh happens at

the first week, all the 131072 bits in this page will be read out, and 394 error bits (from 600 susceptible bits) are corrected in the SSD controller, and then all bits are programmed to a new place. We can see other 130678 bits are read out and then copied back without modifying, which contributes to the most part of the refresh latency. If refreshing only occurs to susceptible cells, the refresh cost will be tremendously reduced, and the endurance of flash memory can be correspondingly extended.

If we want to refresh susceptible cells to address this urgent refresh issue in the 3D NAND flash based storage system, three important aspects should be taken into consideration:

1) How to identify the susceptible cells? Each flash page consists of a group of flash cells, and some cells are not sensitive to retention errors, whereas others are error-prone. An appropriate strategy should be designed to distinguish these two type cells, i.e., how to find the borders of the dominating overlap region (e.g., V_1 and V_5) in Fig. 2?

2) How to refresh these susceptible cells? After identifying susceptible cells, we should determine where to store these susceptible data, and how to organize them in the new place. Furthermore, the mapping table controlled by FTL should be modified to adapt to the new structure.

3) How to read out the full data after refreshing susceptible cells? After refreshing susceptible data, the raw data is composed of two part, the first part is in the old refreshed block, and the second part is in the new page. How to access this data when we want to read them? In addition, how to deal with the increased read cost after refreshing susceptible cells?

3. Proposed Design Solution

To address these aforementioned issues, a novel partial-refresh scheme is proposed in the 3D NAND flash based storage system. We first outline the system architecture of the proposed partial-refresh scheme and then elaborate its major components. At last, we present the overhead analysis.

3.1. Overview

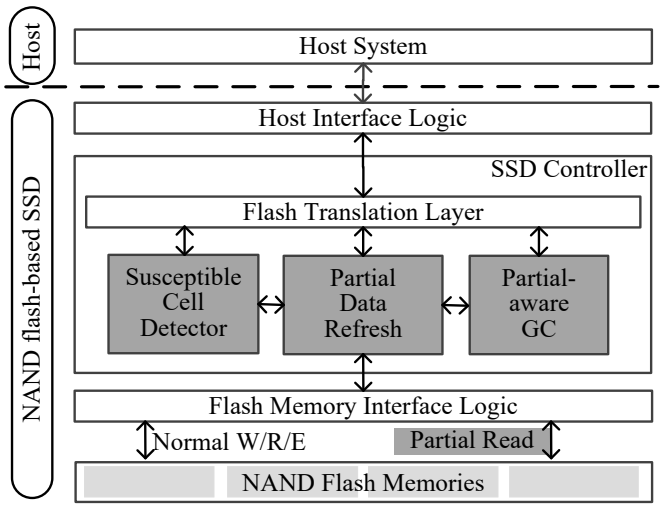

Figure 3: An overview of the partial-refresh scheme.

Fig. 3 presents an overview of the partial-refresh enabled flash-based SSD organization, where partial-refresh is embedded in the SSD controller at the SSD side. The SSD controller is responsible for processing the I/O requests from the host side and managing the address space inside the SSD, maximizing the performance and lifetime, and its major firmware is the flash translation layer. Atop of the SSD controller, the host interface logic as shown in Fig. 3 provides the interface-level compatibility with HDDs, whereas beneath the SSD controller, the flash memory interface logic issues operations (read/write/erase) to the underlying flash memories. The partial-refresh scheme exploits the susceptible cells vulnerabilities in refreshing for both performance and lifetime improvement, and it is composed of these four components: (1) *susceptible cell detector* (SCD), (2) *partial data refresh* (PDR), (3) *partial-aware garbage collection* (PGC), and (4) *partial read* command. SCD identifies cell strength based on the threshold voltage distribution. PDR reduces refresh-induced write latency by exploiting non-susceptible cells retention time. PGC promotes the partial-refreshed pages to the traditional pages, reducing the longer read la-

tency of partial-refreshed pages than that of normal pages. To implement the practical partial-refresh scheme, we also implemented the special command, the partial read operation (*pread*) to access the partial-refreshed data pages. Note that the proposed partial-refresh scheme is primarily designed for 3D flash to address the deteriorated refresh cost, but it can also be used for conventional planar flash memory.

Next, we elaborate the major components in the partial-refresh scheme of 3D NAND flash-based storage system.

3.2. Susceptible Cell Detector

The design goal of *susceptible cell detector* (SCD) is to detect susceptible cells. When decoding data into the SSD controller, SCD leverages LDPC detectability to identify cells that are more vulnerable to the different types of errors.

In the process of LDPC decoding data, three steps are needed, which are memory sensing, data transferring, and LDPC decoding, respectively. In the read process of current LDPC method [20], it invokes the soft-decision memory sensing first with one sensing level (between adjacent flash states), and then transfers the data content to the controller. If the decoding fails, LDPC repeats the three steps with extra one sensing level until decoding succeeds or fails with the full-strength sensing levels (e.g., 7 sensing levels). In each sensing level, a set of preset quantization voltage levels (e.g., V_1, V_2 and others in Fig. 2) is applied. The set of quantization voltage levels selected determines the log-likelihood-ratio (LLR) values in LDPC, which further affects the LDPC error correction capability. Considerable prior works devoted to optimizing the selection strategy of quantization voltage for LDPC-coded NAND flash memory [21, 22, 23]. In our work, SCD focuses on leveraging LDPC detectability to identify cells that are more vulnerable to the different types of errors, and the quantization voltage levels are set by the default LDPC code employed in the 3D NAND flash based storage system. SCD starts directly with the full-strength 7 sensing levels, since the raw bit error rate (RBER) is so high that the refresh

Table 2: An example of the soft-decision sensing with different regions

region	probability (%)	trusted region?	overlap region?	susceptible?
I	23.97	✓		
II	2.55		✓	✓
III	22.55	✓		
IV	2.63		✓	✓
V	22.45	✓		
VI	2.7		✓	✓
VII	23.15	✓		

Note: region represents the divided threshold voltage interval with the sensing levels.

operation is triggered to guarantee data integrity, and the conventional iterative read-retry process is time-consuming.

During the memory sensing stage, the read reference voltage V_{ref} is first set to the quantization voltage level V_1 as illustrated in Fig. 2, and then applying it to the wordline which contains the data to be read, checking the relationship between V_{ref} and the voltage of flash cell V_{th}. If V_{ref} is above V_{th}, it falls into the trusted region of ER state, otherwise, we need further sensing. Then the read reference voltage V_{ref} is further raised to V_2, and checking the relationship between V_{th} and V_{ref} as shown above. Clearly, with this level-by-level manner, we can find out whether the voltage of flash cell falls into a trusted region of one flash state or an overlap region of two adjacent flash states. The flash memory cell is a susceptible cell only if its threshold voltage belongs to an overlap region, otherwise, it is a non-susceptible memory cell. After memory sensing, all the sensed results of one data page will be transferred to the flash controller through flash-to-controller bus.

During LDPC decoding, the LDPC decoder starts to correct the error bits. Since the susceptible cells do not necessarily cause the bit errors, LDPC decoder corrects some of susceptible bits that have gone wrong. Compared with the non-susceptible cells in the trusted region, the probability of the susceptible cells in

the overlap region is very low. According to the region probability of one LDPC sensing example [21] in Table 2, susceptible cell rate can be nearly 7.88%. Thus, susceptible cell rate will be higher than the raw bit error rate. But compared with all cells, susceptible cell rate is still lower.

3.3. Partial Data Refresh

Basic partial data refresh mechanism. To enable the proposed partial-refresh scheme, there are several important modifications in the implementation of the refresh command. The *partial data refresh* (PDR) will periodically remap susceptible cells in each flash page to a new location, otherwise it accumulates many retention errors. As shown in Fig. 4, the operational flow of the PDR scheme is like below: (1) When the flash refresh occurs at each refresh interval, the SSD's firmware selects a victim block that needs to be refreshed. (2) After that, valid pages are read out and corrected all the errors by ECC engine in the SSD controller. (3) With the assist of SCD, all the susceptible cells are detected. We store the logical page number (LPN) of each victim page, and the susceptible data and its offset within the victim data page into a small memory, named as the *virtual shadow memory* (VSM), controlled by SSD controller. To prevent susceptible data loss, we leverage capacitor-backed RAM in SSD [24] as the VSM. The SSD's firmware maintains more susceptible cells to merge a new flash write, such that susceptible cells can be cached durably and mapped to a better choice. (4) A certain number of susceptible data and its offset in the VSM forms a virtual cache line, in the purpose of generating a new flash write. The virtual cache line is a certain amount of data stored in the VSM, and it is exactly the same size as one flash page, e.g., 16KB. When a flash write occurs, a new free flash block is selected as a *combination block* to service susceptible data, and one virtual cache line in the VSM is sequentially programmed to the free page on the *combination block*. Thus, the copied data during the refresh operation, are extremely reduced. We recommend that PDR selects the *combination block* from other idle flash planes (or even other idle flash chips, exploiting chip idleness), for the purpose of improving the latter partial read (*pread*) performance.

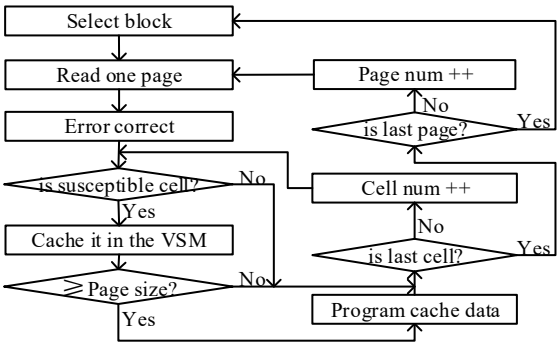

Figure 4: Flow diagram of the partial-refresh-aware design.

Adaptive partial data refresh mechanism. In the above basic partial data refresh mechanism, the fewer refresh-induced programmed data to the SSD, tends to prolong the SSD lifetime. This is because SSD specifies a fixed of P/E cycle as an endurance limit, and with the less program counts to flash memory, the remaining lifetime in the SSD is longer. On the other hand, after refreshing data pages partially, the combination pages are used to store susceptible data and its offset, and at the same time, the previous refreshed page is still valid. Although the percentage of susceptible data is much lower, the basic partial data refresh mechanism may leave less free pages available for new writes and thus invoke GC frequently, in the long term. Hence, the main idea of the adaptive partial-refresh mechanism is to adapt the partial-refresh to the available space of an SSD. Conservatively, when the available free space drops to a certain level, e.g., 20% of the SSD, we must disable partial-refresh, and SSD operates normally with the conventional refresh operation. Only when SSD have sufficient free storage space, we perform the partial-refresh scheme, affecting GC performance only slightly.

Data structure in the VSM. In the context of the partial-refresh scheme, susceptible data are stored in the VSM. Compared with non-susceptible bits, susceptible bits are extremely sparse. In order to store them efficiently, each susceptible bit information is composed of the offset address within each codeword, and the bit value, as shown in Fig. 5. Let N denote the page length, and

M is the codeword length. Then for one page, VSM contains no more than $\frac{N}{M}$ codeword, and we can achieve $log_2 M+1$ bits field for each susceptible bit. To set a fixed size of the offset address, a redundancy entry of the first bit in each codeword is employed, e.g. 001 and 000 entry, as illustrated in Fig. 5. Note that no matter the first bit in each codeword is erroneous or not, we record it in the VSM, thus offset address length of susceptible bits can be set as the fixed size, $log_2 M$. The 2th bit is stored as 010, where 01 is the offset address value and 0 is the susceptible bit value corrected by ECC engine. And the 7th bit, which is also susceptible cell, is stored as 101, because offset address from this second codeword is 10, and the bit value is still 1.

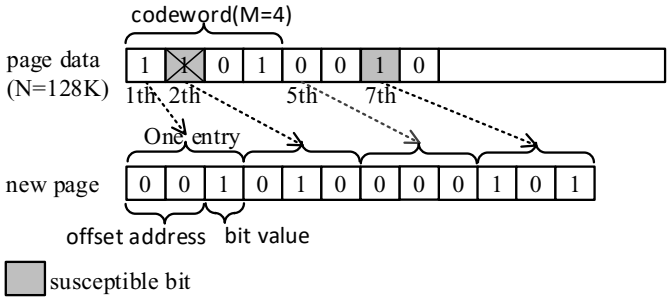

Figure 5: Illustration of the susceptible bits storage scheme.

Since a certain amount of susceptible data and its offset in the VSM forms a virtual cache line to generate a new flash write, the susceptible data of one victim page may across the virtual cache line in the VSM, generating two or more flash write operations. To prevent the susceptible data of the victim page from storing in more than one combination page, we delay these susceptible data for a while into the next virtual cache line, and choose other susceptible data where its data page has lower susceptible data rate than that of the victim page. And towards the end of the virtual cache line, if we can not find the susceptible data with the appropriate amount of storage space in the VSM, we fill them with zeros directly.

The partial-refresh aware mapping table. To enable the partial-refresh scheme, we design an enhanced mapping mechanism. When a new flash write

operation is generated, susceptible data and its offset in the virtual cache line of the VSM are encoded with LDPC codes. FTL then determines the physical address of this encoded data based on the cached mapping table. In the partial-refresh scheme, the PPN is still valid, and the susceptible data are programmed to a new location.

To this end, we maintain a cached primary mapping table and a cached secondary mapping table, as shown in Fig. 6. If data is without the partial-refresh operation, the cached primary mapping table maps an LPN to a PPN, as the traditional mapping strategy (1-to-1 mapping). If data is a partial-refreshed data, the cached primary mapping table additionally maps a PPN to an additional physical page number (APN), with the help of the LPN information in the VSM (1-to-2 mapping). And each entry is indexed by APN and has two fields, {APN, *offset*}. The *offset* tracks the offset address within each combination page. Note that, if we continuously perform the partial-refresh operation on the partial-refreshed data, the same LPN will be written to more than 2 physical pages. Instead of developing some expensive methods to prevent 1-to-n address mapping (n $\geqslant$ 3), we take a simple method which flips the refresh type. That is, if one block has been partial-refreshed or was the combination block, at the refresh interval it would service a conventional refresh command.

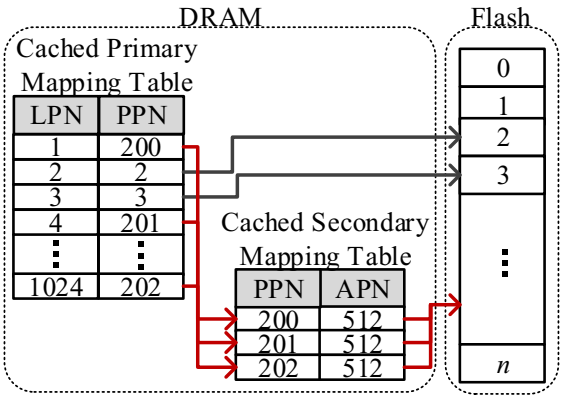

Figure 6: Illustration of the enhanced mapping table.

We add the cached secondary mapping table in the mapping table, which increases DRAM capacity for maintaining larger mapping table, and the theoretical and empirical research demonstrates that the effects of large mapping table on storage performance are negative. But in the practical design of SSD firmware, on-demand map loading [25] or hybrid mapping [26] schemes are widely used to reduce the mapping table size. Therefore, the entry count in the mapping table is not going to be large, and the storage overhead introduced in the mapping table is negligible.

3.4. Partial-read Operation

The *pread* command. The partial-refreshed data is composed of two parts: the original old data page in the partial-refreshed block, and the remapped susceptible data in the combination page. Partial-refresh-supporting SSD must handle this read command differently from normal reads, thereby, we need to implement a novel special read command. Existing stroage interface of SSD, such as SATA or SAS [27], supports the user-defined command to further optimize the storage system. In this work, we implemented the *pread* operation by utilizing the legacy read command. As shown in Fig. 7, the special *pread* command is composed of two sub-reads, which is enabled by utilizing the normal page read command to access the original old data page in the partial-refreshed block (first sub-read), and the partial-page read command to access the susceptible data from the combination block (second sub-read). The partial-page read command designed by the previous work [28], is used to read only part of a page, not an entire page. Both the page read operation and partial-page read operation are combined into one *pread* command. Such *pread* command sequence is practical, because some existing advance commands, e.g., multi-plane read operations, are similarly issued. The access finish time of *pread* is determined by the access finish time of the slowest sub-read, even if another sub-read can finish early. After accessing data from two locations, susceptible data is integrated into the raw data content. It checks the *offset* value in the secondary mapping table to acquire the needed susceptible data in the combination page. After

that, we check the offset value in each susceptible entry to substitute raw old data with the susceptible bit value. Then data are processed by the decoding engine, and the *pread* operation completes. Note that the *pread* command can be implemented in the storage interface of SSD that supports the user-defined command, thus, no hardware changes are required.

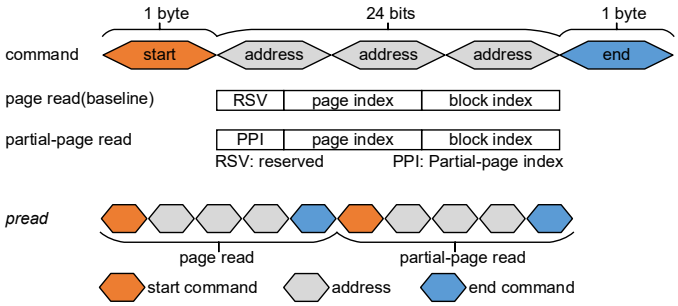

Figure 7: The *pread* command format.

Separating hot/cold data. Since reading partial-refreshed data tends to introduce the longer read latency than reading traditional data, the data hotness identification strategy is implemented to enhance refresh performance. We categorize data hotness according to their read ratio after programming, i.e., data that most statistical information is read, is treated as read-hot one [29]. Note that the hotness identification technique is orthogonal to our proposed designs and can be replaced with other state-of-the-art hotness metrics. During each refresh interval, the valid pages are read out, only if one page is regarded as read-hot, it invokes the traditional refresh strategy. Otherwise, the proposed partial-refresh design invokes to exploit the non-susceptible cells. Therefore, with the data hotness information, we can prevent partial-refreshing the frequently-read data pages, which will introduce severe performance degradation.

3.5. Partial-aware Garbage Collection

Since above partial-refresh scheme contributes to different page types, e.g., normal or partial-refreshed pages, a new partial-aware GC strategy is proposed to enhance the default greedy GC:

To select a victim block for GC, we consider both the number of valid pages and their refresh type. We select the block that has fewer valid partial-refreshed pages as the victim block with priority. This is because, if copying the same number of valid pages in two types of flash blocks, the number of partial-refreshed pages is smaller, then the GC latency is shorter. When page copying, we should also consider their refresh type. For the victim block with valid partial-refreshed pages, the valid data come from two different positions, which are both the current victim block and another combination block. These partial-refreshed pages still accumulate retention errors after partial refreshing. To prevent these pages from suffering too many errors, we promote these valid partial-refreshed pages to traditional pages using the *pread* operation, and set the partial susceptible data in the corresponding combination page to invalid. Once all the susceptible bits in one combination page are invalid, the state of the combination page becomes invalid.

3.6. Overhead Analysis

Partial-refresh, while improving flash-based storage system performance and prolonging the endurance of flash chip, introduces storage overhead and firmware overhead.

1) Storage Overhead. Partial-refresh needs following extra storage space to save the metadata:

- Partial-refresh keeps a VSM in memory as shown in Section 3.3 to temporarily store some susceptible data, in the purpose of generating a new flash write. For partially refreshing each victim page, we store its LPN and all the susceptible data in the VSM. The 32-bit LPN is used for updating relevant information in the enhanced mapping table, where 32-bit page address is sufficiently large for a 128GB SSD with 16KB 3D flash pages. And each susceptible data in the VSM, i.e. a susceptible entry, is composed of a $log_2 M$-bit offset address within each ECC codeword and a 1-bit value, when M denotes the ECC codeword length. If n is the

number of susceptible data, the maximum storage overhead for storing the affected susceptible data in VSM should be $n \times (32+log_2 M +1)$-bit. But in practice, in the purpose of generating a new flash write, the size of all cached susceptible entries in the VSM should be larger than one page size. Therefore, we conservatively set the size of the VSM to several page sizes, e.g. 4×16KB.

- Partial-refresh keeps an enhanced mapping table, as shown in Fig. 6, to identify the partial-refreshed data in the internal DRAM of an SSD. In the enhanced mapping table, we first need a 1-bit flag to differentiate partial-refreshed data and non-partial-refreshed data. Then, in the 32-bit page address, we need a 32-bit flag in the secondary mapping table to indicate the APN address, which is the same size as the PPN. And, we also need a 20-bit flag to track the offset address within each combination page, which is sufficiently large for a 16KB page. Since in the practical design of the enhanced mapping table, we leverage the on-demand map loading policy to reduce the mapping table, and the storage overhead introduced in the enhanced mapping table is negligible.

2) Firmware Overhead. We need the processes involved in the susceptible cell detector, partial data refresh and partial-aware garbage collection. The overhead of these simple processes is quite negligible.

4. Analysis and Experiments

4.1. Methodology

We evaluate partial-refresh on a trace-driven SSD simulator, SSDsim [30]. We simulate a 128GB capacity 32-layer 3D NAND flash-based SSD, and its key parameters are listed in Table 3. The simulated 3D flash has four channels, and each of which consists of one flash chip. And in each flash chip, there are total 2192 blocks, where each block is composed of 1024 16KB pages. The latency for read, program and erase operation is 75 μs, 1,050 μs, and 10,000 μs, respectively.

Table 3: Key parameters of 3D flash configuration

Parameter	Value	Parameter	Value
capacity	128GB	page size	16KB
channel	4	OP ratio	7%
chip per channel	1	WL	Dynamic
die per chip	1	DA scheme	CWDP
plane per die	4	sensing time	75 μs
block per plane	548	program time	1050 μs
page per block	1024	erase time	10000 μs
P/E	4K	GC threshold	10%

The maximum P/E cycle count is specified at 4K for the simulated 3D NAND flash based SSD.

Partial-refresh is simulated by mapping several susceptible data from different logical pages to a physical page that has to be read and written. To implement the partial-refresh scheme, the average susceptible ratio (the fraction of bytes that are susceptible in one page) is set to be 7.88% [21]. And we disable partial-refresh when the ratio of available free blocks to total blocks in the simulated SSD drops to 20%.

The workloads. We using 9 real-world enterprise server traces at Microsoft Research Cambridge to evaluate the proposed scheme [31]. In order to trigger refresh commands, we perform a linear extrapolation to shorten the data retention time according to Table 1, since the tracing duration is about one week.

Schemes for comparison. In the subsequent experiments, the following schemes are implemented:

- `FCR.` The most recent work FCR that optimizes refresh operation [8]. We adopt this scheme as the baseline scheme. All the experimental results were normalized to FCR.
- `EC-Cache.` This is a read optimization scheme that is close to our de-

sign, which caches the positions and valid values of detected errors with the assist of LDPC [32]. Both EC-Cache and our proposed partial-refresh scheme exploit the LDPC powerful detectability for performance improvement.

- `PR`. This is the proposed partial-refresh scheme for both performance and lifetime improvement of 3D flash-based SSDs, by leveraging LDPC detectability to identify cells that are more vulnerable to errors.

4.2. Performance Improvement

Refresh latency: Fig. 8 compares the refresh latency for different schemes. From the Fig. 8, PR significantly reduces the average refresh latency by 28.21% over `FCR`. The refresh performance improvement comes from: (1) During each partial-refresh interval, SCD starts sensing data with full-strength memory sensing levels. But conventional the read process invokes the memory sensing with one sensing level, and higher RBER leads to more iterative read-retry process. Therefore, the read response time in the partial-refresh is reduced; (2) Due to tremendously reduced write latency in each partial-refresh interval, the write response time is reduced. In summary, `PR` achieves the largest write latency reduction, i.e., 30.95% in *rsrch*, and the smallest reduction, i.e., 24.71% in *src2*. Note that, since `EC-Cache` only optimize the read operation for the data that are frequently read, it has the comparable refresh latency quality as that in `FCR`. Overall, these results clearly demonstrate the effectiveness of `PR` in improving refresh performance.

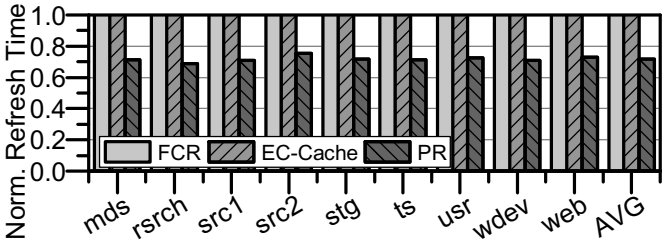

Figure 8: Comparing the refresh latency under different workload traces.

To fully understand the performance improvement in the proposed partial-refresh scheme, Fig. 9 reports the partial ratio in the partial-refresh scheme, where the partial ratio is defined as the fraction of page counts that are partial-refreshed in the total refreshed page counts. As we can see that the refresh latency reduction compared to `FCR` is larger when the partial-refreshed page ratio is higher. This is because with the higher partial-refreshed page ratio, the more write time in the refresh interval is reduced, thereby improving better refresh performance. For example, the partial-refreshed page ratio is 24.37% in *rsrch*, while `PR` achieves the maximum refresh latency reduction by 30.95% over `FCR`.

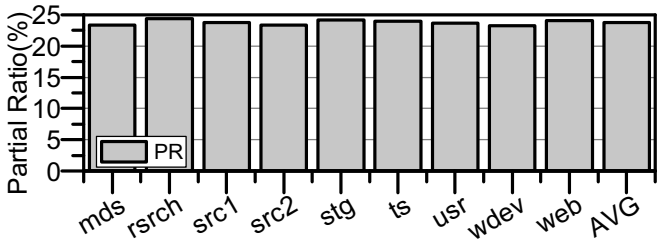

Figure 9: Percentage of partial-refreshed pages in the partial-refresh scheme.

Average response time: To demonstrate partial-refresh impact on the I/O performance, we also compare the average response time as shown in Fig. 10. From Fig. 10, `PR` achieves response time reduction by 21.39% when compared with `FCR`. The average response time is affected by both the read response time (or called the read latency) and the write response time (or called the write/program latency). The performance improvement in `PR` comes from the reduced program latency during the partial-refresh stage. Meanwhile, the LDPC decoding time is also reduced in `PR` when compared with `FCR`, thereby read latency is also reduced. Since `EC-Cache` has the comparable refresh latency as that in `FCR`, it also has the comparable average response time as that in baseline `FCR`.

Write latency: To verify that `PR` does reduce the program latency effectively, Fig. 11 compares the average write response time (read performance compari-

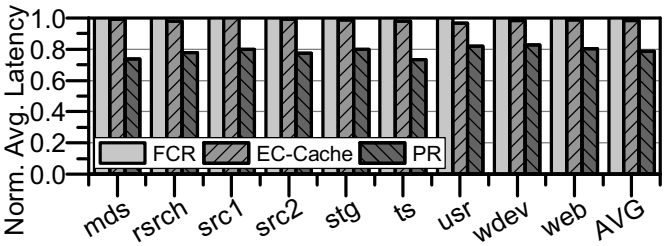

Figure 10: The average response time comparison.

son is explained later). From Fig. 11, PR achieves 26.38% average write response time reduction, compared to FCR. As can be observed, our proposal can successfully reduce the write latency, compared to the baseline FCR proposal. This is because, in each partial-refresh interval, we only program the susceptible data to new pages, leading to the reduced program latency dramatically. While in the traditional refresh operation, all the valid data (regardless of susceptible data or the large percentage of non-susceptible data) should be refreshed to new pages.

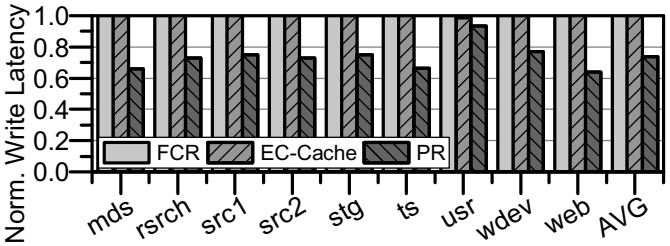

Figure 11: The average write response time comparison.

4.3. Lifetime Improvement

Lifetime: Fig. 12 compares the the lifetime of flash chip when adopting different schemes. On average, PR achieves about 4.64% flash lifetime improvement over FCR. This is because PR reduces the number of write operations when partial refreshing the blocks with valid data. The more valid pages in partially refreshing the workload has, the larger the lifetime improvement is. Since all valid

pages in partial refreshing are rewriting in the separate combination blocks, PR shows slightly better lifetime improvement than FCR.

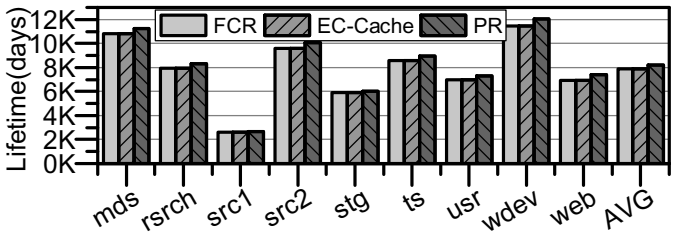

Figure 12: Comparing flash chip lifetime.

Write amplification: Write amplification is another factor that impacts the lifetime of flash chip. With the larger write amplification factor, the lifetime of flash chip is shorter. Fig. 13 shows the effects on write amplification comparing among FCR, EC-Cache and PR. On average, PR reduces the average write amplification by 3.80% over FCR. PR achieves the largest reduction, i.e., 6.40% in *web*, and the smallest reduction, i.e., 1.89% in *src1*. On the one hand, since PR reduces the amount of data moved drastically, the number of writes is largely reduced. On the other hand, since the fewer write counts on flash chips, it tends to trigger GC with lower frequency. Therefore, the number of GC-induced page copying is also reduced, and the write amplification is further reduced in PR.

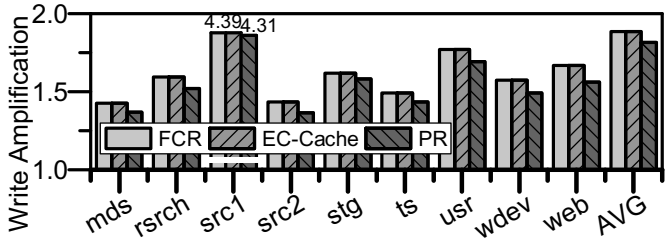

Figure 13: Write amplification for each benchmark.

Erase counts: We next evaluate the number of erase operations over all the workloads when adopting different schemes. Flash memory manufacturers specify a fixed of P/E cycle as an endurance limit. If one scheme consumes more

erase counts after runtime, the remaining lifetime in the flash-based device is shorter. As shown in Fig. 14, PR reduces the number of erase operations by 5.42%. EC-Cache has the comparable erase count quality as that in FCR, because it only improves the read performance for the frequently read data. In the partial refresh period of the PR scheme, all the preceding valid data content are still valid, erase operation is not triggered, therefore the erase count is reduced.

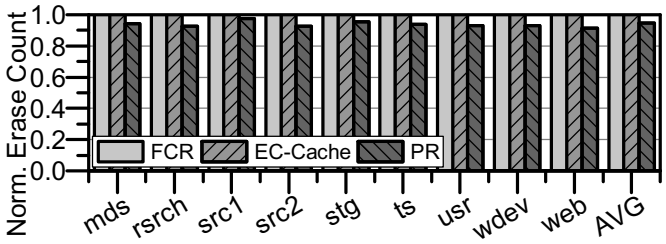

Figure 14: The number of erase comparison.

4.4. Read Overhead Analysis

At last, we studied the overhead of PR. Although PR technique shows its better performance and lifetime improvement than the traditional refresh method, it introduces one main overhead: additional read counts due to the *pread* command, which might reduce the read performance.

Read latency: We report the average read response time comparison in Fig. 15. Compared with FCR, the read overhead in PR is low, i.e. 6.68% on average. The comparable read performance comes from the data hotness identify strategy is also implemented in PR, the frequently-read data pages in the selected refresh block will not invoke the partial-refresh design, which prevents introducing severe read performance degradation. Compared with the traditional FCR, EC-Cache achieves significant read performance improvement. This is because EC-Cache is used to optimize the read operation for the data that are frequently read.

To fully understand the read performance degradation in the proposed partial-refresh scheme, Fig. 16 reports the percentage of the *pread* operations in the

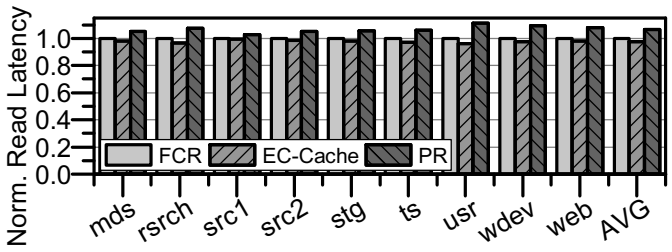

Figure 15: The average read response time comparison.

partial-refresh scheme. By comparing the results in Fig. 15 and Fig. 16, we observed that when the percentage of the *pread* operations is higher, then the read latency increment compared to `FCR` is larger. For example, for *usr* the *pread* ratio is 6.01% in `PR`, while compared to `FCR`, `PR` achieves the maximum read latency increment by 11.10%. This is because the more *pread* operations tend to have larger queuing latency, which increases the average read response time.

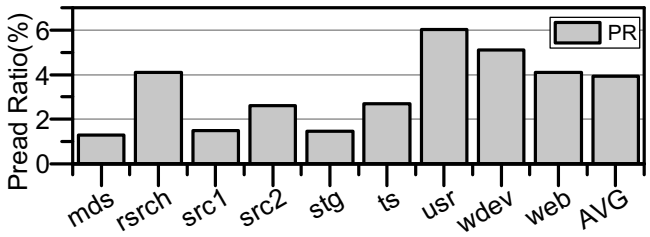

Figure 16: Percentage of the *pread* operations in the partial-refresh scheme.

5. Limitations and Discussions

In this section, we now summarize the limitations of the proposed partial-refresh scheme and explore some directions for the future work. First, partial-refresh depends on LDPC, hence partial-refresh-aware SSDs should adopt LDPC as the default ECC scheme. Fortunately, modern SSD products widely deploy the powerful LDPC to improve the reliability of SSDs [20, 33]. Second, the enhanced mapping table in the partial-refresh-aware SSDs introduce extra storage

cost, i.e. consuming more DRAM capacity. As described previously, we can further leverage on-demand map loading or hybrid mapping schemes to reduce the mapping table size. Finally, partial-refresh introduces the additional read counts due to the *pread* command (as evaluated in 4.4), which requires future work as below.

In our current work, we implemented a read hotness identification strategy, and only when one data page is identified as read cold, we invoked the proposed partial-refresh design. If the read-hot data is misidentified as the read-cold data, these partial-refreshed data pages can be promoted to the traditional data pages, which can be improved for better read performance. Also, we can leverage the current increased capacity of DRAM in the modern SSDs [34] to selectively cache some partial-refreshed or combination pages, improving better read performance. We leave the effective design for future work.

6. Related Work

Our work leverages LDPC detectability to check out vulnerable bits and the proposed refresh minimization scheme is designed to provide significant gains in flash performance and lifetime. Our work is inspired and motivated by many related works.

Big block related proposals. With the recent 3D NAND technical innovations, the capacity of each 3D flash block has been significantly increased which causes a serious performance degradation at the storage system level. Many prior works focus on this "*big block*" problem and show that it is much more severe in 3D flash compared to 2D flash. Partial-erase or sub-block erase operation for 3D NAND flash has been proposed in [14, 35] to boost system performance. The partial GC mechanisms [27, 36] conducted in the context of planar NAND flash are proposed to partition GC into multiple operation partitions in order to avoid channel resource conflicts and reduce GC-induced latency. This motivates us to investigate the deteriorated refresh-induced latency and lifetime reduction in 3D NAND flash-based storage system, and we develop a new fine-grained

partial-refresh mechanism to address the urgent refresh issue in the 3D NAND flash memory.

Refresh related proposals. Many prior works have focused on flash refresh operation optimization. Some researches [10, 11, 12, 9] reduce the refresh frequency according to different specific metrics, e.g. temperature, data writing frequency, improving the lifetime of the flash memory storage. Meanwhile, some researches reduce the refresh operation time at each refresh interval. Cai *et al.* [8] optimized the refresh process itself by selectively re-programming data in-place, which reduces the traditional remapping-based refresh induced latency. However, recharging a 3D flash page is unsubstantial since a programmed 3D flash page is subject to more adjacent disturb errors. Unlike previous works, we based on the early retention loss phenomenon in 3D flash, and proposed a more fine-grained lightweight data refresh mechanism to reduce refresh cost.

Exploiting LDPC detectability. Recently, Liu *et al.* proposed EC-Cache[32], a read optimization that caches the positions and valid values of errors detected with the assist of LDPC. EC-Cache records the detected error bits which have errors, however, we record susceptible bits that include both detected error bits and these bits that are going to be wrong. With these susceptible bits, partial-refresh exploits the susceptible cells vulnerabilities in refreshing for performance and lifetime improvement.

Other performance and lifetime optimization proposals in CPU subsystems. Considerable research achievements have been conducted to improve performance and/or lifetime in CPU subsystems. Prior works proposed many different scheduling algorithms in CPU subsystems to improve the reliability for modern MPSoC systems [37, 38, 39, 40], and enhance the high-performance MPSoC systems [41, 42]. These proposals, however, were designed for CPU subsystems, and they addressed specific issues e.g., resources scheduling, task scheduling. Flash memory, however, is quite different from CPU, and it is widely adopted in the modern secondary storage subsystems as a storage device. Since flash memory require refresh operations to guarantee long-term data integrity, in this paper we designed a novel refresh minimization scheme to improve the

I/O performance and extends the lifetime of flash memory.

7. Conclusion

This work proposed partial-refresh (PR), a novel refresh minimization scheme designed for 3D flash memory in cyber-physical systems. By leveraging LDPC detectability to check out bits that are more vulnerable to errors, a partial-refresh operation is implemented to copy these susceptible bits to new pages. A partial-read operation is implemented to handle the read requests that access the partial-refreshed data, differently from normal read requests. Furthermore, an efficient implementation of partial-aware garbage collection approach is presented. Experimental results show that, a PR-aware FTL improves the refresh performance by 28.2% and extends the lifetime of 3D flash by 4.6%.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China [grant number 61902136, 61932010 and 61672423], and by Fundamental Research Funds for the Central Universities [grant number 2019kfyXJJS090], and by National Science Foundation of the United States [grant number CCF-1718080], and by the National Key Research and Development Program of China [grant number 2016YFB1000303].

References

[1] Wang, Xiaokang and Yang, Laurence T and Xie, Xia and Jin, Jirong and Deen, M Jamal, A cloud-edge computing framework for cyber-physical-social services, IEEE Communications Magazine 55 (11) (2017) 80–85.

[2] Wang, Xiaokang and Yang, Laurence T and Kuang, Liwei and Liu, Xingang and Zhang, Qingxia and Deen, M Jamal, A tensor-based big-data-driven routing recommendation approach for heterogeneous networks, IEEE Network 33 (1) (2019) 64–69.

[3] Wang, Xiaokang and Yang, Laurence T and Li, Hongguo and Lin, Man and Han, Jianjun and Apduhan, Bernady O, NQA: A nested anti-collision algorithm for RFID systems, ACM Transactions on Embedded Computing Systems 18 (4) (2019) 32.

[4] Manderscheid, Martin and Weiss, Gereon and Knorr, Rudi, Verification of network end-to-end latencies for adaptive ethernet-based cyber-physical systems, Journal of Systems Architecture 88 (2018) 23–32.

[5] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, O. Mutlu, Improving 3d nand flash memory lifetime by tolerating early retention loss and process variation, Proc. POMACS 2 (3) (2018) 37.

[6] R. Micheloni, 3d flash memories (2016) 29–30.

[7] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, O. Mutlu, Data retention in mlc nand flash memory: Characterization, optimization, and recovery, in: Proc. HPCA, 2015, pp. 551–563.

[8] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, K. Mai, Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime, in: Proc. ICCD, 2012, pp. 94–101.

[9] R.-S. Liu, C.-L. Yang, W. Wu, Optimizing nand flash-based ssds via retention relaxation, Target 11 (10) (2012) 00.

[10] M. Seif, E. Farjallah, F. Badets, E. Chabchoub, C. Layer, J.-M. Armani, F. Joffre, C. Anghel, L. Dilillo, V. Gherman, Refresh frequency reduction of data stored in ssds based on a-timer and timestamps, in: Proc. IEEE Test Symposium, 2017, pp. 1–6.

[11] Y. Luo, Y. Cai, S. Ghose, J. Choi, O. Mutlu, Warm: Improving nand flash memory lifetime with write-hotness aware retention management, in: Proc. MSST, 2015, pp. 1–14.

[12] Y. Di, L. Shi, K. Wu, C. J. Xue, Exploiting process variation for retention induced refresh minimization on flash memory, in: Proc. DATE, 2016, pp. 391–396.

[13] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, T.-W. Kuo, Garbage collection and wear leveling for flash memory: Past and future, in: Proc. SMARTCOMP, 2014, pp. 66–73.

[14] C.-y. Liu, J. Kotra, M. Jung, M. Kandemir, Pen: design and evaluation of partial-erase for 3d nand-based high density ssds, in: Proc. FAST, 2018, pp. 67–82.

[15] T. Tanaka, M. Helm, T. Vali, R. Ghodsi, K. Kawai, J.-K. Park, S. Yamada, F. Pan, Y. Einaga, A. Ghalam, et al., 7.7 a 768gb 3b/cell 3d-floating-gate nand flash memory, in: Proc. ISSCC, 2016, pp. 142–144.

[16] Q. Xiong, F. Wu, Z. Lu, Y. Zhu, Y. Zhou, Y. Chu, C. Xie, P. Huang, Characterizing 3d floating gate nand flash, in: Proc. SIGMETRICES, 2017, pp. 31–32.

[17] Y. Cai, O. Mutlu, E. F. Haratsch, K. Mai, Program interference in mlc nand flash memory: Characterization, modeling, and mitigation, in: Proc. ICCD, IEEE, 2013, pp. 123–130.

[18] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, L. R. Nevill, Bit error rate in nand flash memories, in: Proc. IRPS, IEEE, 2008, pp. 9–19.

[19] Y. Cai, E. F. Haratsch, O. Mutlu, K. Mai, Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling, in: Proc. DATE, 2013, pp. 1285–1290.

[20] J. Cui, Y. Zhang, W. Wu, J. Yang, Y. Wang, J. Huang, Dlv: Exploiting device level latency variations for performance improvement on flash memory storage systems, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 37 (8) (2018) 1546–1559.

[21] G. Dong, Y. Zou, T. Zhang, Reducing data transfer latency of nand flash memory with soft-decision sensing, in: Proc. ICC, 2012, pp. 7024–7028.

[22] Liu, Wenjie and Han, Guojun and He, Ruiquan and Fang, Yi and Cai, Guofa, Dynamic-Reference-Voltage-based Detection Algorithm for LDPC-Coded NAND Flash Memory, in: International Conference on Wireless Communications and Signal Processing, IEEE, 2018, pp. 1–5.

[23] Micheloni, Rino and Marelli, Alessia and Onufryk, Peter Z, System and method with reference voltage partitioning for low density parity check decoding, US Patent 9,235,467 (january 2016).

[24] T. Pott, Supercapacitors have the power to save you from data loss, http://www.theregister.co.uk/2014/09/24/storage_supercapacitors/, [Online; accessed 24-September-2014] (2014).

[25] A. Gupta, Y. Kim, B. Urgaonkar, DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings, Vol. 44, ACM, 2009.

[26] D. Ma, J. Feng, G. Li, Lazyftl: a page-level flash translation layer optimized for nand flash memory, in: Proc. SIGMOD, 2011, pp. 1–12.

[27] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, M. T. Kandemir, Hios: A host interface i/o scheduler for solid state disks, in: ACM SIGARCH Computer Architecture News, Vol. 42, 2014, pp. 289–300.

[28] C.-Y. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, C. R. Das, Soml read: Rethinking the read operation granularity of 3d nand ssds, in: Proc. ASPLOS, 2019, pp. 955–969.

[29] J. Cui, W. Wu, X. Zhang, J. Huang, Y. Wang, Exploiting latency variation for access conflict reduction of nand flash memory, in: Proc. MSST, 2016, pp. 1–7.

[30] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, S. Zhang, Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity, in: Proc. ICS, 2011, pp. 96–107.

[31] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, A. Rowstron, Migrating server storage to ssds: analysis of tradeoffs, in: Proc. EuroSys, 2009, pp. 145–158.

[32] R.-S. Liu, M.-Y. Chuang, C.-L. Yang, C.-H. Li, K.-C. Ho, H.-P. Li, Ec-cache: Exploiting error locality to optimize ldpc in nand flash-based ssds, in: Proc. DAC, 2014, pp. 1–6.

[33] Zhao, Kai and Zhao, Wenzhe and Sun, Hongbin and Zhang, Xiaodong and Zheng, Nanning and Zhang, Tong, LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives, in: Presented as part of the 11th USENIX Conference on File and Storage Technologies, 2013, pp. 243–256.

[34] Kang, Mincheol and Lee, Wonyoung and Kim, Soontae, Subpage-aware solid state drive for improving lifetime and performance, IEEE Transactions on Computers 67 (10) (2018) 1492–1505.

[35] T.-Y. Chen, Y.-H. Chang, C.-C. Ho, S.-H. Chen, Enabling sub-blocks erase management to boost the performance of 3d nand flash memory, in: Proc. DAC, 2016, p. 92.

[36] M. Jung, R. Prabhakar, M. T. Kandemir, Taking garbage collection overheads off the critical path in ssds, in: Proc. Distributed Systems Platforms and Open Distributed Processing, 2012, pp. 164–186.

[37] Zhou, Junlong and Hu, X Sharon and Ma, Yue and Sun, Jin and Wei, Tongquan and Hu, Shiyan, Improving availability of multicore real-time systems suffering both permanent and transient faults, IEEE Transactions on Computers.

[38] Zhou, Junlong and Sun, Jin and Zhou, Xiumin and Wei, Tongquan and Chen, Mingsong and Hu, Shiyan and Hu, Xiaobo Sharon, Resource management for improving soft-error and lifetime reliability of real-time MPSoCs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[39] Zhou, Junlong and Wei, Tongquan and Chen, Mingsong and Yan, Jianming and Hu, Xiaobo Sharon and Ma, Yue, Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (8) (2015) 1269–1282.

[40] Liu, Jing and Wei, Mengxue and Hu, Wei and Xu, Xin and Ouyang, Aijia, Task scheduling with fault-tolerance in real-time heterogeneous systems, Journal of Systems Architecture 90 (2018) 23–33.

[41] Yang, Tao and Deng, Qingxu and Sun, Lei, Building real-time parallel task systems on multi-cores: A hierarchical scheduling approach, Journal of Systems Architecture 92 (2019) 1–11.

[42] Temuçin, Hüseyin and İmre, Kayhan M, Scheduling computation and communication on a software-defined photonic Network-on-Chip architecture for high-performance real-time systems, Journal of Systems Architecture 90 (2018) 54–71.