# Near-Free Lifetime Extension for 3D NAND Flash via Opportunistic Self-Healing

Tianyu Ren[1], Qiao Li[2], Yina Lv[1], Min Ye[3], Nan Guan[1], and Chun Jason Xue[4]

[1]Department of Computer Science, City University of Hong Kong
[2]School of Informatics, Xiamen University
[3]YEESTOR Microelectronics Co., Ltd
[4]Mohamed bin Zayed University of Artificial Intelligence

***Abstract*—3D NAND flash memories are the dominant storage media in modern data centers due to their high performance, large storage capacity, and low power consumption. However, the lifetime of flash memory has decreased as technology scaling advances. Recent work has revealed that the number of achievable program/erase (P/E) cycles of flash blocks is related to the dwell time between two adjacent erase operations. A longer dwell time can lead to higher achievable P/E cycles and, therefore, a longer lifetime for flash memories. This paper found that the achievable P/E cycles would increase when flash blocks endure uneven dwell time distribution. Based on this observation, this paper presents an opportunistic self-healing method to extend the lifetime of flash memory. By maintaining two groups with unequal block counts, namely *Active Group* and *Healing Group*, the proposed method creates an imbalance in erase operation distribution. The *Active Group* undergoes more frequent erase operations, resulting in shorter dwell time, while the *Healing Group* experiences longer dwell time. Periodically, the roles of the two groups are switched based on the *Active Group*'s partitioning ratio. This role switching ensures that each block experiences both short and long dwell time periods, leading to an uneven dwell time distribution that magnifies the self-healing effect. The evaluation shows that the proposed method can improve the flash lifetime by 19.3% and 13.2% on average with near-free overheads, compared with the baseline and the related work, respectively.**

## I. Introduction

3D NAND flash memory has been developed with characteristics of high performance, large capacity, and low energy consumption, which gradually replaces planar flash memory and has become mainstream media for many storage systems. However, with the multi-bit-per-cell technique and layer-stacking architecture, the lifetime of flash memory has significantly reduced, which hinders its further deployment [1]. Fig. 1 shows the trend of the P/E cycle limit that the modern 3D NAND flash memory can achieve, provided by the flash vendors. The transition from SLC (1-bit-per-cell) flash memory to PLC (5-bit-per-cell) flash memory has resulted in a significant reduction in the P/E cycle limit, decreasing from several thousand to several hundred. As technology continues to advance, the P/E cycles of the memory are expected to undergo further reduction. Consequently, addressing the lifetime degradation issue of high-density flash memory becomes crucial to extending their operational lifetime in both consumer and enterprise environments.

To tackle the lifetime degradation issue of flash memory, multiple approaches have worked on heat-accelerated healing and prolonged dwell time of blocks. These efforts aim to facilitate self-healing mechanisms that can effectively extend the P/E cycle limit of flash memory. We termed the extended P/E cycle limit as the achievable P/E cycle. First, the heat-accelerated healing method applies heating operations on flash blocks for an aggressive healing process [4]–[8]. However, heating operations, which require hardware support, are exceptionally time-consuming and must relocate the data in heated blocks before heating. Cui et al. [9] proposed to avoid unnecessary heating operations by partitioning the blocks into groups. Only the blocks exceeding their P/E cycle limit and experiencing short dwell time need to be heated. While this method can reduce the time required for the heating operation to some extent, hardware modifications are still necessary. Second, the dwell time-based self-healing method revealed that the achievable P/E cycle of flash blocks is related to the dwell time between two adjacent erase operations during garbage collection (GC). Extending the dwell time of a block can improve the block's achievable P/E cycle [10], [11]. Lee et al. [11] presented the relationship between the achievable P/E cycles and dwell time with a set of parameters based on a 20nm MLC flash memory, and proposed to throttle the write requests for achieving a better SSD's lifetime extension, with the cost of poor write performance. However, an accurate model with proper parameters to present such a relationship on high-density 3D NAND flash memory is still missing. Besides, no prior work has explored the self-healing effect to improve the lifetime of endurance-limited 3D NAND flash memory.

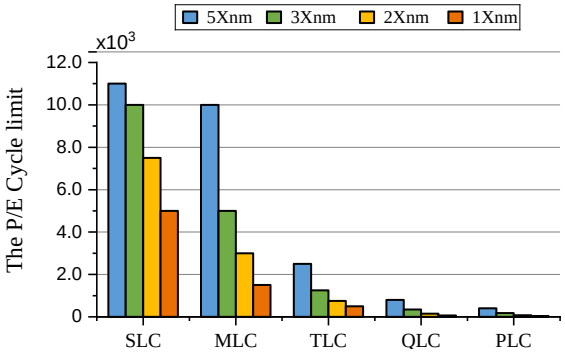



Fig. 1. The P/E cycle limit degraded with the increased cell density [2], [3].

In this paper, we first conduct experiments on real 3D TLC

flash chips to study the relationship between the number of achievable P/E cycles and dwell time, filling the void of critical parameters in the model for high-density 3D NAND flash memory. With the model, we further observed that *uneven distribution* of dwell time, during the lifetime of a block, is beneficial to the self-healing effect of flash blocks. The number of achievable P/E cycles in a block becomes larger if the block endures long and short dwell time during its lifetime, compared to enduring even distribution of dwell time. Based on this observation, we propose an opportunistic self-healing method for flash memory that aims to prolong the lifetime by leveraging the self-healing benefits brought by the uneven distribution of dwell time. The proposed approach maintains two groups with an unequal number of blocks: the *Active Group* and the *Healing Group*. The *Active Group* is subjected to a higher frequency of erase operations, resulting in a shorter dwell time for its blocks. In contrast, *Healing Group* experiences longer dwell time from fewer erase operations. After a predetermined period, which depends on the partitioning ratio of *Active Group*, the roles of the two groups will be switched, with some blocks in the *Healing Group* taking on the role of *Active Group*. At the same time, all the blocks from the previous *Active Group* will be switched to the *Healing Group*. This periodic role switching ensures that each block undergoes both short and long dwell time periods, creating an uneven distribution of dwell time. By alternating high and low erase operation frequencies, the proposed method enhances the self-healing effect of flash blocks, leading to an extended lifetime of flash memory. Notably, our approach incurs near-free overheads, making it an attractive solution for improving the lifetime of flash-based storage systems.

In summary, this paper makes the following contributions:

- We show that there is a great potential to improve the overall self-healing effect based on the relationship between the achievable P/E cycles and the block dwell time, where the model is updated by conducting experiments on the newly released 3D TLC NAND flash chip;
- We propose an opportunistic self-healing method to improve flash lifetime by maintaining two groups with uneven numbers of flash blocks, thus achieving an improved self-healing effect;
- Experiments are conducted on a widely used simulator with TLC flash extensions. Evaluation results show that the proposed method can effectively improve the flash lifetime with near-free overhead.

## II. Background

### A. Basics of NAND Flash Memory

3D NAND flash memory is widely used as the storage medium in the solid-state drive (SSD), due to its high performance, large capacity, and low power consumption. The SSD includes a flash memory array and a controller. The flash memory array is composed of several chips, and the chips are connected by I/O buses. Inside the flash chip, there are thousands of blocks with a three-dimensional (3D) stacking architecture. Fig. 2 shows the internal structure of a flash

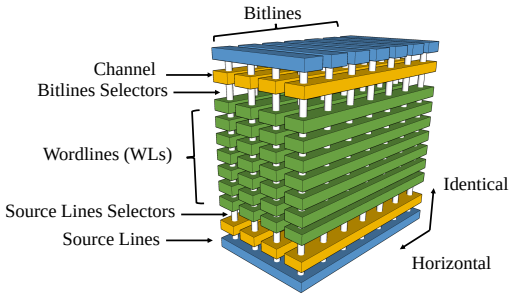



Fig. 2. The internal structure of a 3D NAND flash memory block.

block. Cells in the horizontal direction form a wordline, while cells in identical directions compose a string. Data in 3D NAND flash memory uses threshold voltages to represent the bit information. The number of bits in a storage cell has increased from one bit to five bits. For example, a cell that can maintain 2-bit data is called a multi-level cell (MLC). To cater to the needs of high density, triple-level cell (TLC) and quad-level cell (QLC) have become the mainstream 3D NAND flash memory type. There are three types of bits in a TLC NAND flash memory: least significant bit (LSB), center significant bit (CSB), and most significant bit (MSB). In 3D NAND flash memory, read, program, and erase are three basic operations. The basic unit of read and program operations is a page, but the erase operation reclaims a block for each time.

There are several management components integrated into the SSD controller to manage internal activities, such as address mapping, garbage collection (GC), wear leveling (WL), and so on. The address mapping is used to translate the logical address to the physical address of data stored in the flash array. The GC scheme is designed for space reclamation when the remaining free space is not enough. Once the number of free blocks is lower than a predefined threshold, the GC is triggered. When GC moves valid data from one block to another, it can cause uneven wearing on the memory cells. This is because some cells may be written to or erased more frequently than others, depending on the pattern of data usage and the way that garbage collection is implemented. The WL scheme is responsible for managing such uneven wearing among different flash blocks. WL algorithms aim to evenly distribute the program and erase cycles across all blocks, which can prevent individual blocks from wearing out earlier than others and becoming bad blocks. This helps extend the lifetime of the flash memory and enables even the wearing of NAND flash blocks.

Combining a well-designed WL algorithm with an effective GC strategy is essential for optimizing the lifetime of flash-based storage systems. The current GC policy is the greedy GC [12], [13] that selects the victim block with the largest number of invalid pages in a plane to erase, which can free up more space in the SSD, and then the performance can be improved. Despite the benefits, the greedy GC also has potential drawbacks. For example, the greedy GC may prioritize freeing up space quickly without considering the long-term effects of the GC process, such as the fragmentation of data. Furthermore, the greedy GC may cause valid page migration, where valid data need to be rewritten during each GC process.

When the number of migrating valid data is high, it means that the SSD has performed more additional write operations than necessary, which can lead to a decrease in the lifetime of NAND flash memory and overall performance. Therefore, how to improve the efficiency of the GC process in modern high-density flash memory has become a well-studied topic [13]–[16]. In particular, multi-stream technology [14], [16] is one of the effective approaches for reducing valid page migration, which divides data into streams based on data update characteristics. Each stream is responsible for writing data with the same update frequency. In this way, data with the same update frequency can be gathered into the same block so that they expire at the same or similar time. However, the lifetime of a flash block is primarily limited by the wear on the flash block itself as electrons continuously charge and leak. Additional data writing during GC and WL is the culprit that speeds up the flash wear. Fortunately, flash blocks can be self-healed, which can alleviate the wearing. In this paper, we focus on how to optimize the flash lifetime with the self-healing effect.

### *B. The Healing Effect of Heating on Flash Blocks*

Flash blocks are capable of storing data but have a P/E cycle limit. To improve the endurance and extend the P/E cycle limit of NAND flash memory, various techniques have been developed. One approach is to apply heating operations on flash blocks, which requires redesigning the internal memory's structure by adding the mini heater. Heating operations involve exposing the memory cells to an exceeded high temperature (e.g., 800 °C [17]) for a short period of time to repair the damage caused by repeated P/E cycles on flash cells, which can extend flash cells' lifetime. The rationale is that the high temperatures during the heating operation can activate certain self-healing effects in the flash cells, such as dissipating trapped charges. Because of the physical characteristics, there is a limit on the number of heating operations that can be performed on each flash block. Until the flash memory cells reach their maximum number of heating operations, the P/E cycle limit, which is fixed and specified by the vendor, can be extended to the maximum achievable P/E cycle after experiencing a self-healing effect. The duration and temperature of the heating operation must be carefully controlled to prevent excessive stress on the flash cells, which can cause additional damage and degrade the flash's performance. Besides the cost of an additional internal heater in NAND flash memory, valid data must be removed from a free block before heating a worn-out block, which results in additional write operations and a significantly increased number of valid data migrations.

### *C. Self-Healing Effect of Dwell Time*

Accumulating P/E cycles trap extra electrons in the tunnel oxide of flash cells, which shifts the voltage states of cells in the direction of higher value and deteriorates the endurance of flash memory. Thus, the raw bit error rates (RBERs) of stored data are significantly increased, and the lifetime of flash memory is limited to some specific number of P/E cycles. After a period of retention time, the trapped charge slowly dissipates, which is called flash self-healing [10], [17]. The idle time between two successive P/E cycles is referred to as the **Dwell Time (DT)**, which can help free electrons trapped in the oxide layer [4], [18], [19]. Eq. (1) shows the relationship between dwell time and the threshold voltage shift difference ($\triangle \mathrm{V}_{th}$) caused by self-healing effect [11]:

$$\triangle V_{th} = (A_{it} \times N_{pe}^{0.62} + B_{ot} \times N_{pe}^{0.3})(1 - C \times ln(\frac{DT}{t_0})) \quad (1)$$

where N means the number of P/E cycles, DT denotes the dwell time, and the constant C represents the healing efficiency [11]. $t_0$ typically accounts for the impact of the DT, while the $A_{it}$ and $B_{ot}$ regulate the influence from the P/E cycles. As inferred from Eq. (1), the $\triangle \mathrm{V}_{th}$ is proportional to the dwell time, and thus, a longer dwell time can achieve a better self-healing effect.

This paper presents a novel approach that performs a proactively self-healing procedure in 3D NAND flash memory. By carefully designing the GC process to consciously control the dwell time for each block throughout the memory's lifetime, we can improve endurance and reliability without incurring additional costs.

## III. MOTIVATION

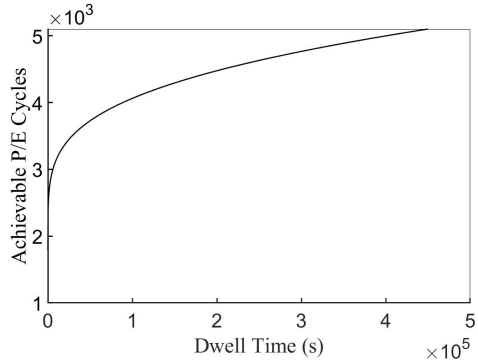



Fig. 3. The relationship between the achievable P/E cycles (ranging from 0 to 5k P/E cycles) and the dwell time (ranging from 0 to $5\times10^5$ seconds).

The dwell-time-driven self-healing process facilitates the gradual release of trapped electrons from the oxide layer, leading to the self-healing effect, which prolongs the lifetime of the flash blocks, surpassing the P/E cycle limit set by the flash memory manufacturer. Previous works have studied the relationship between the achievable P/E cycles ($\mathrm{PE_a}$) and the dwell time (DT), which is demonstrated in Eq.(2) [4], [9], [20]:

$$PE_a = \frac{RBER_{ECC} - RBER_{init}}{k \times ln(1 + \frac{DR}{t_0 + C \times DT})} - g \quad (2)$$

where $g$ and $t_0$ are constant, and $k$ and $C$ are coefficients, which are set to $3.359\times10^{-1}$, $7.848\times10^{-1}$ seconds, $1.213\times10^{-7}$ and $4.877\times10^{-1}$ , respectively. The $k$ and $g$ control the impact of RBER on the $\mathrm{PE_a}$, and $C$ and $t_0$ are the same parameter in Eq. (1). The $\mathrm{RBER_{init}}$ is presented as $\epsilon + \alpha\ \mathrm{PE_a}^m$ [21], where $\epsilon$, $\alpha$ and m is $1.484\times10^{-1}$, -1.597 $\times10^{-1}$and $-8.2\times10^{-3}$, respectively. The $\mathrm{RBER_{ECC}}$, which is the correction ability of error correction code (ECC) in the BCH ECC module, is set to 160 bits/16KB [22]. We assume that the data retention time for stored data is three months, which is required by enterprise SSD [23]. Hence DR is set to $7.776\times10^6$ seconds.

The above parameters, except the $\text{RBER}_{\text{ECC}}$ and DR, are obtained by testing on the latest 3D TLC charge trap NAND flash memory from one vendor. This flash chip has 2224 blocks and 178 layers in each block. To perform program and erase operations on the actual chip, we utilize the YEESTOR 9082HC flash memory testing platform [24]. We selected three different blocks with the same setting to avoid experimental bias. Before writing random data, the tested blocks undergo 2K P/E cycles with varying dwell time, such as 1s, 16s, 20s, 30s, 40s, 80s, 85s, and 90s. Subsequently, the tested blocks are subjected to accelerated data retention under 100°C about 96 minutes to simulate one year's retention time under real working environment temperatures (i.e., 25°C) by using the Arrhenius equation, showing in Eq. (3), to calculate the acceleration factor [25], [26].

$$t_{high} = t_{room} \times exp(\frac{E}{K} \times (\frac{1}{T_{high}} - \frac{1}{T_{room}})) \tag{3}$$

where the Boltzmann constant $K$ and activation energy constant $E$ are set as 8.62×10-5eV/K and 1.1eV, respectively.

Eq. (2) is graphically interpreted in Fig. 3, which is a log-like function, presenting the relationship between the achievable P/E cycles and the dwell time. The achievable P/E cycles increase with the increase of dwell time. In particular, we observe that uneven distribution of dwell time during the lifetime of a block can achieve higher achievable P/E cycles. We take the following cases to illustrate the above observation. Assume that there are $N$ pieces of data written into $M$ blocks in a time period of $T$, where $T$ is set to three months. $M$ represents an SSD's capacity. Note that the value of $N$ should be greater than the value of $M$. Otherwise, no GC operations will be triggered. To simplify the assumption model, the size of each data piece is equal to the capacity of a block. If we evenly distribute all the data among blocks, each block will write $\frac{N}{M}$ data, so the average dwell time of each block is $\frac{(T \times M)}{N}$, called $\text{DT}_{ave}$. On the other hand, if we divide the block into two parts, one takes up $T_{blk}$, and the other is *1-*$T_{blk}$. Similarly, we also divide the data into two parts: $T_c$ and *1-*$T_c$. $T_c$'s data is written into the blocks of part $T_{blk}$, and vice versa. Thus, the dwell time ($\text{DT}_{healing}$, $\text{DT}_{active}$) for each part during the entire healing procedure could be calculated as follows:

$$DT_{healing} = \frac{T_{blk}}{T_c} \times \frac{T \times M}{N} \tag{4}$$

$$DT_{active} = \frac{1 - T_{blk}}{1 - T_c} \times \frac{T \times M}{N} \tag{5}$$

Suppose $DT_{ave}$ is 63072 seconds, so the lifetime of NAND flash memory can roughly reach 3833 P/E cycles, according to Eq. (2). Now, when $T_{blk}$ is set to $\frac{7}{10}$ and $T_c$ is set to $\frac{1}{10}$, the dwell time of the first part ($DT_{healing}$) will be 441504 seconds, and we call this part as the *Healing Group*. For the second part, the dwell time ($DT_{active}$) will about be 21024 seconds, and we call this part the *Active Group*. Therefore, blocks in the *Healing Group* will obtain 5085 achievable P/E cycles, and the achievable P/E cycles of the blocks of *Active Group* will be 3387 P/E cycles. In this case, by dividing the blocks into two groups and storing different amounts of data in each group, the average achievable P/E cycles for each block will be 4236, which is 10.5% higher than evenly writing the data into all blocks. The other case is that we assign $\frac{7}{10}$ of blocks to the *Healing Group* and handle $\frac{1}{2}$ of data, while $\frac{3}{10}$ of blocks are in the *Active Group* to serve the remaining data. As a result, we can achieve only 3804 achievable P/E cycles, which is even 0.7% lower than the case of evenly distributing the data across all the blocks. These two cases indicate that appropriately partitioning flash blocks to accommodate varying amounts of data can substantially improve the self-healing process of flash blocks. This, in turn, can lead to an overall improvement in the lifetime of SSDs. The relationship between the achievable P/E cycle and the dwell time remains logarithmic [9]. Therefore, the aforementioned discussion is applicable to different flash chips, where the only difference lies in the detailed parameter values. To ensure that the *Healing Group* undergoes fewer GC

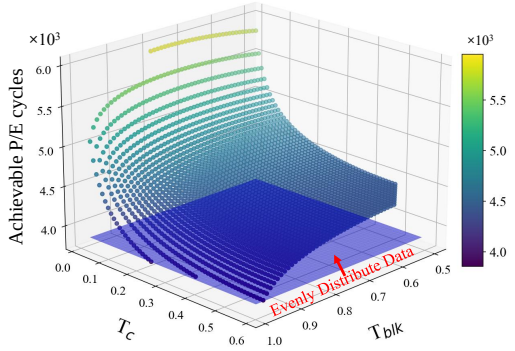



Fig. 4. The analysis of thresholds $T_{blk}$(ranging from 0.5 to 1), which is the ratio of block count in *Healing Group*, and $T_c$ (ranging from 0 to 0.5), which is the ratio of the GC count on *Healing Group* (The range of the y-axis is from 3K to 6K P/E cycles).

operations than the *Active Group*, the value range of $T_c$ is set between 0 and 0.5. Additionally, our switch is based on the number of blocks in the *Active Group*. To ensure that a block has a longer duration in the *Healing Group*, the value range of $T_{blk}$ is set between 0.5 and 1. Since Eq. (2) is a concave function, we should avoid the selection of $T_{blk}$ and $T_c$ to make the average dwell time of the *Healing Group* and the *Active Group* less than or equal to the case that all the data are evenly distributed. Based on the same assumption mentioned above, we plot Fig. 4 to guide parameter selection. It can be observed that, at the same value of $T_{blk}$, a smaller value of $T_c$ leads to a greater enhancement in the lifetime of flash memory, compared to the translucent plane in the middle representing the achievable P/E cycles when data is evenly allocated to all blocks. Fig. 4 can provide valuable guidance for selecting threshold values during the implementation of the opportunistic self-healing method. In summary, without the requirement for additional hardware, we can simply group the blocks and control the amount of data written to each group. Then, by periodically switching the blocks in these two groups, blocks can experience a long and short dwell-time period. By creating an uneven distribution of dwell time among blocks' lifetime, we can extend the lifetime of the flash memory. Therefore, we need to improve the existing GC strategy and deliberately control the data writes.

## IV. DESIGN

### A. Overview

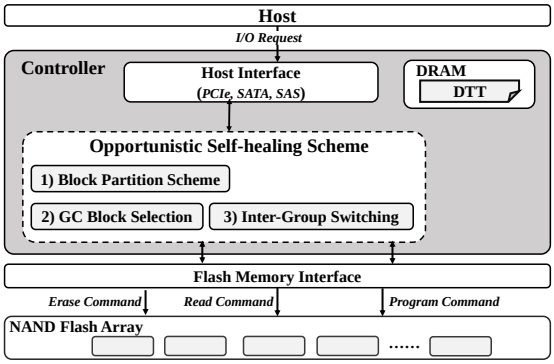



Fig. 5. The architecture of storage system with the opportunistic self-healing, including three main components: 1) Block Partition Scheme, 2) GC Block Selection, and 3) Inter-Group Switching.

In this section, an opportunistic self-healing method is proposed to extend the lifetime of NAND flash-based storage systems. Fig. 5 shows the architecture of the opportunistic self-healing method, including three main components: 1) *Block Partition Scheme*; 2) *GC Block Selection*; 3) *Inter-Group Switching*. We propose to maintain two distinct groups with an unequal number of blocks. The key idea is to intentionally create an unevenness in the erase operation distribution between these two groups. The group that frequently undergoes erase operations is called the *Active Group*, whereas the group that experiences fewer erase operations is known as the *Healing Group*. This uneven distribution of erase operations will result in the blocks in *Active Group* enduring shorter dwell time, while the blocks in *Healing Group* endure longer dwell time. After a predetermined period, which depends on the partitioning ratio of *Active Group*, the roles of the two groups will be switched, with some blocks in the *Healing Group* taking on the role of *Active Group*. In tandem with this, all the blocks that constituted the previous *Active Group* will be reassigned to the *Healing Group*. This periodic switching of roles ensures that each block experiences a period with short dwell time and the remaining period with long dwell time, achieving the goal of uneven distributions of dwell time. This process enhances the self-healing effect of flash blocks, thus prolonging the lifetime with near-free overheads. Meanwhile, the dwell time of blocks is important statistical information in our approach. Therefore, we propose an efficient and space-saving method called *Dwell Time based Block Management* for recording blocks' dwell time.

To realize the idea, several challenges should be addressed. **First**, we should correctly record the dwell time for each block without introducing excessive storage overhead. **Second**, how to assign blocks to the *Healing Group* and the *Active Group* remains a challenge. A simple solution is that all blocks in each plane can be separated into two groups based on the hotness of service requests. One is responsible for serving the hot write data, and the other one can handle the cold write data. The number of blocks in each group is determined by the proportion of cold and hot data from upper-level applications, but the setting of block quantities between the two groups can considerably affect the flash lifetime, as illustrated by the second case presented in Section III. It is necessary to deliberately control the number of blocks in two groups. **Third**, for GC victim block selection, we should consider both the number of invalid pages within a block, which determines the GC efficiency, and the dwell time of flash blocks, which affects the self-healing effects. The existing GC strategy is no longer suitable for our approach, and it needs improvement. To uneven the dwell time of blocks, a new strategy that selects victim blocks from different groups should be introduced. **Finally**, the conventional WL algorithm is designed to wear evenly across the entire lifetime of blocks, which is against the idea of our proposed method. We aim to unevenly distribute wear on blocks at arbitrary points during their lifetime, but ultimately, each block will experience the same degree of wear. Therefore, our proposed method should apply a new mechanism to control the wear on blocks. In the following, the detailed design will be discussed to address these challenges.

### B. Dwell Time based Block Management

In this section, we introduce a dwell-time based block management method to facilitate the implementation of the opportunistic self-healing method. Within the dwell time of a flash block, it undergoes three distinct statuses: free block, open block, and data block. There is a conversion relationship among these three block statuses. First, when a free block is selected and ready for data writing, it becomes an open block. Second, if we continue to write data into the open block until there are no remaining free pages, the open block will become a data block and we need to find another free block as the open block. The data block is then erased and becomes a free block for future writing services. As introduced in Section II, the dwell time of a block is defined as the time interval between two adjacent erase operations. Thus, the dwell time of a block, denoted as $DT$, encompasses the time spent in the free block status, the open block status, and the data block status.

The dwell time information of each block is recorded in a table, named dwell time table (DTT), which is stored in the DRAM of SSD's controller, as shown in Fig. 5. Each entry in the DTT includes the block ID and the last erase time of this block. To ensure the accurate tracking of the $DT$ for each block, the recorded information needs to be updated during each GC process. Assume that a data block (e.g., the block ID is 1) has participated in the $i-1$ erase operation during the GC process. Then, the time point of this erase operation is labeled as $erase_{i-1}$ that has been recorded in the DTT. Once a block is erased at the $i$th erase operation, the time point of the $i$th erase operation is referred to as the $erase_i$, and will be updated in the DTT. Therefore, given the time points of two erase operations, we can calculate the $i$th DT of a block. For instance, if a block with ID 1 was erased at time point 8, the last erase time of block 1 in the DTT would be updated to 8, which means that the next dwell time of block 1 starts to accumulate, and this value will be updated until the block is selected for the

next erase operation. Such dwell time can be calculated as the interval between the time point of the current erase operation and the last erase time point recorded in the DTT. For example, if the time point of the current erase operation is 64 while the last erase time is 8, then the current dwell time of block 1 can be calculated as 64-8=56. At the same time, the last erase time of block 1 will be updated as 64 in the DTT. Gradually, we can separately collect the average dwell time for each block in two distinct groups: the average long dwell time when the block is in the *Healing Group* and the average short dwell time when the block is in the *Active Group*. According to Eq. (2), we can individually calculate the achievable P/E cycles for each block within the two groups. Ultimately, each block's final achievable P/E cycle is determined by averaging the two achievable P/E cycles.

As discussed in Section III, the uneven distribution of dwell time can prolong the achievable P/E cycles of a block. The dwell time information is used as a significant indicator to reveal the self-healing effect of a block. In the following, we will provide a comprehensive description of our proposed opportunistic self-healing method, achieving flash lifetime optimization by controlling block dwell time.

### C. Opportunistic Self-Healing

In this section, we propose an *opportunistic self-healing* scheme to optimize flash lifetime. The basic idea is to partition blocks into two distinct groups: the *Healing Group* and the *Active Group*. During each GC process, we strategically select the victim block for erasure from either group. This approach aims to achieve a longer dwell time for blocks in the *Healing Group*, and a shorter dwell time for blocks in the *Active Group*. To achieve this goal, we have introduced three schemes to manage flash blocks and garbage collection. **Firstly**, to achieve the desired partition ratio between two groups, we design a *Block Partition Scheme* that ensures the block count ratio between the two groups remains within a specified threshold. **Secondly**, a new GC process called *GC Block Selection*, is designed to select the victim block from each group, considering both the dwell time and the number of invalid pages. This scheme makes a tradeoff between the GC efficiency and the self-healing effect of certain blocks. **Thirdly**, we propose *Inter-group Switching* to ensure not only uneven wear distribution across each block at arbitrary points but also achieve uniform wear levels among all blocks over time. The above strategies of the *opportunistic self-healing* scheme are described in detail in the following.

*1) Block Partition Scheme:* We aim to make some blocks experience short dwell time periods with more frequent erase operations, while other blocks with long dwell time periods and less frequent erase operations. A practical approach to naturally distinguish between these two types of blocks without compromising SSD performance is considering the data hotness since the hot and cold data have different update frequencies, directly affecting the GC process frequency on the data blocks. Inspired by this, separating cold and hot data naturally divides the blocks into two distinct groups, termed *Healing Group* and *Active Group*, as illustrated in Fig. 6. The blocks in the *Healing Group* allocated with cold data have longer dwell time,

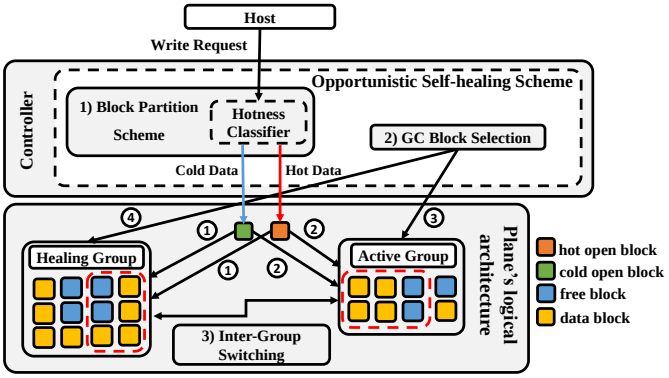



Fig. 6. The opportunistic self-healing method overview. Case ① and ② represent the two open blocks that can be assigned to which group based on $T_{blk}$. Case ③ and ④ represent which group to run GC based on $T_c$.

while blocks in the *Active Group* that store hot data experience shorter dwell time. The ratio of cold and hot data depends on the access behaviors of upper-layer applications, leading to variability in the number of blocks within each group. However, not all partitioning ratios result in the desired extension of the block's lifetime, as described by the second case in Section III. Hence, regulating the number of blocks within these two groups becomes necessary. In summary, we initially divide blocks into different groups based on the hotness of the data. As the volume of data requests from the upper layer increases, we re-assign data to the other group for incoming requests once the ratio between the two groups deviates from our expectations.

This paper adopts the frequency of data updates [15] to indicate the hotness of data. We use a counter to record the update frequency of incoming write requests. Data with update frequencies exceeding a specified threshold is classified as hot data, while data with frequencies below this threshold is classified as cold data [27]. The proposed method incorporates multi-stream capability, allowing two blocks to simultaneously stay in the open status in each plane [28], where one block is for cold data and one is for hot data. By employing different blocks for storing cold and hot data, distinct invalidation frequencies are observed in these two kinds of blocks, and the logical capacity is split into two distinct parts.

In summary, we first place the incoming cold or hot data from the upper layer into the respective open block, with cold data going into the open block primarily designated for cold data and hot data into the open block mainly designated for hot data. Subsequently, the open block that stores the cold data will be assigned to the *Healing Group*, whereas the open block storing hot data will be assigned to the *Active Group*. With each write request, we check the number of blocks within both groups. Once the block ratio between these two groups deviates from the pre-defined threshold, we may reassign the current open block to another group. We introduce a threshold, $T_{blk}$, to determine the partition ratio of the *Healing Group*, as shown in Eq. (6).

$$\#\ Blk\ in\ HG = T_{blk} \times Total\ Blk\ Number \tag{6}$$

where the value of $T_{blk}$ ranges from 0.5 to 1, and the *# Blk in HG* means the number of blocks in the *Healing Group*. As shown in Fig. 6, initially, we identify the hotness of the write request from the host through the *Hotness Classifier*. Then, we distribute data to one of two open blocks based on their hotness. If the open block is for hot data, we assign it to the *Active Group*. Conversely, if the open block is for cold data, we will assign it to the *Healing Group*. After the block finishes the data writing and becomes a data block, it stays in the corresponding group. Once the number of blocks in the *Healing Group* falls below the threshold (less than $T_{blk}$ times the total number of blocks in the plane, like Case ① in Fig. 6), we assign the open block to the *Healing Group*. On the contrary, if the number of blocks in the *Healing Group* exceeds $T_{blk}$ times the total number of blocks in the plane, like Case ② in Fig. 6, the open block will be assigned to the *Active Group*. With the above strategy, each block will be assigned to either *Active Group* or *Healing Group*, and the ratio between them approximately reaches the desired setting after the initial assignment.

*2) GC Block Selection:* To further uneven the dwell time of blocks in these two groups, it is necessary to redesign the GC victim selection during the runtime. When a block is frequently erased, its dwell time is shortened, while blocks that are not frequently selected for GC operations have a longer dwell time. Therefore, it is crucial to meticulously design our GC victim block selection strategy to distribute different GC frequencies across two groups.

*GC Block Selection* for *opportunistic self-healing* considers both dwell time and the invalid page counts of flash blocks. Our objective is to conduct more GC operations on blocks in the *Active Group*, thereby shortening their dwell time. Conversely, we reduce the chance of selecting blocks from the *Healing Group* as victim blocks, enabling them to have longer dwell time. Based on this principle, we control the probability of the GC operations occurring in the two groups by setting a threshold ($T_c$), which satisfies the following relationship:

$$\textit{GC Count in HG} <= T_c \times \textit{Total GC Count} \tag{7}$$

where $T_c$ varies between 0 and 0.5, and *GC Count in HG* means the number of GC operations that happened in the *Healing Group*. We prioritize selecting blocks ($ABlks$) from the *Active Group* for erasure, like Case ③ in Fig. 6, as shown in **Line 20** of **Algorithm 1**. Yet, if the number of erasures on blocks ($HBlks$) from the *Healing Group* falls below a certain value ($T_c$ times the total number of GC operations), like Case ④ in Fig. 6, the GC operation will select the victim block from the *Healing Group* for reclamation, as shown in **Line 25** of **Algorithm 1**.

The selection strategy for victim blocks within a group follows the principle of greedy GC, where we prioritize blocks with the highest number of invalid pages for erasure. Nonetheless, when we choose a victim block to be erased, the block with the highest number of invalid pages within one group might have much fewer invalid pages than the block with the highest number of invalid pages in the whole plane. The reason is that both groups may contain blocks that predominantly store cold data with lower update frequencies, resulting in a smaller number of invalid pages in these blocks. Selecting such blocks could lead to a considerable degradation in GC efficiency and introduce more GC counts. Therefore, when performing GC operations on both groups, we will implement an ***invalid page checkpoint mechanism***. Each time, we compare the number of invalid pages in the selected victim block ($InPage$) with the highest number of invalid pages across the entire plane ($MaxInPage$). As demonstrated from **Line 10** to **Line 11 of Algorithm 1**, if $InPage$ is less than $T_i \times MaxInPage$ in the same plane, where $T_i$ is a value between 0 and 1, we consider that erasing this victim block may lead to low GC efficiency and substantial data migration of valid page. Thus, we do not select this victim healing block to erase but re-select a new victim block with the highest number of invalid pages from another group. In this case, the newly selected block is the block with the maximum number of invalid pages across the entire plane.

The setting of $T_i$ can largely impact the effectiveness of our approach. If the value of $T_i$ is set too small, the valid data migration can not be well addressed. On the other hand, if the value is set too large, the GC strategy is close to the traditional greedy GC strategy, with little control over the distribution of dwell time across blocks. Therefore, it is crucial to carefully select the value of the $T_i$ threshold.

*3) Inter-Group Switching:* Following a period of opportunistic self-healing, the blocks in the *Healing Group* have a longer

**Algorithm 1** The Process of the GC Block Selection.

```
1: function FVicBlk(Blks, BlkCnt, T)
2:     InPage ← −∞;
3:     VicBlk ← −1;
4:     for i ← 0 to BlkCnt − 1 do
5:         if Blks[i].BlkInPage > InPage then
6:             InPage ← Blks[i].BlkInPage;
7:             VicBlk ← i;
8:         end if
9:     end for
10:    if InPage < T then
11:        return −1;
12:    else
13:        return VicBlk;
14:    end if
15: end function
16: procedure GCBlkSelect( )
17:    Initialize variable T_c, T_i, TotalGCCnt, ABlks, HBlks,
       MaxInPage
18:    T ← T_i × MaxInPage;
19:    if GC on HG > T_c × TotalGCCnt then
20:        VicBlk ← FVicBlk(ABlks, ABlkNum, T);
21:        if Victim = −1 then
22:            VicBlk ← FVicBlk(HBlks, HBlkNum, T);
23:        end if
24:    else
25:        VicBlk ← FVicBlk(HBlks, HBlkNum, T);
26:        if VicBlk = −1 then
27:            VicBlk ← FVicBlk(ABlks, ABlkNum, T);
28:        end if
29:    end if
30:    return VicBlk;
31: end procedure
```

dwell time due to fewer GC operations. At the same time, the blocks in the *Active Group* have experienced extensively frequent GC operations with shorter dwell time. To realize the uneven distribution of the dwell time for each block throughout its lifetime, we propose to periodically switch the blocks in the *Active Group* with the blocks in the *Healing Group*. In the following, we will provide a detailed description of how to perform the switch between these two groups.

We use the same threshold for partitioning groups, $T_{blk}$, to control the time for role switching. When the average erase count of all blocks within a plane reaches $T_{blk}$ × P/E cycle limit, a switch is triggered. Firstly, all the existing blocks in the *Active Group* will be assigned to the *Healing Group*. Then, we select (1-$T_{blk}$) multiplied by the total number of blocks from the *Healing Group* as the new *Active Group*. During the switch, the blocks in the *Healing Group* that have been previously assigned to the *Active Group* will not be reassigned to the *Active Group* again. This way, when the SSD approaches the end of its lifetime, all blocks will experience an equal number of P/E cycles, without specialized WL strategies. In terms of the traditional WL, all the blocks experience a similar degree of wear throughout their entire lifetime. However, our proposal aims to create an uneven dwell time distribution of each block during its lifetime, by repeatedly swapping blocks from *Healing Group* to *Active Group*. Therefore, in this work, we replace the traditional WL algorithm with the *Inter-Group Switching* mechanism.

### D. Discussion and Analysis of Implementation Overhead

The proposed method incurs two additional costs. First is the number of valid page migrations, resulting from the potentially lower GC efficiency. Second is the storage overhead of DDT. Therefore, it is crucial to strike a balance between the benefits of achieving higher P/E cycles and the costs associated with the number of valid page migrations. In our approach, during the GC process, we select the victim block within a group based on the maximum number of invalid pages it contains. However, this victim block may not necessarily have the highest number of invalid pages among all blocks in the entire plane. If we were to erase this block, it would lead to a minor reclamation of space and the migration of a larger number of valid pages. This is especially evident when the GC occurs in the *Healing Group*, as most of the blocks in this group are storing cold data. Therefore, overheads caused by inefficient GC operations can be compensated by the benefits of the self-healing effect. We introduce *invalid page checkpoint mechanism* for controlling invalid pages in the victim block, which greatly reduces the negative impact caused by the number of valid data migrations. We have conducted the corresponding sensitivity studies in Section V.B.4) to present the effects. Compared to previous work, we do not need to add additional hardware to perform passive self-healing operations on the memory cell. We only need to redesign the GC strategy to enable the memory cell to perform the self-healing process actively. Therefore, there is no additional firmware overhead since our proposed method is implemented inside the SSD controller. To realize the *Dwell Time based Block Management*, we need to use a table to record the last erase time of each block. For a 128GB SSD, such storage overhead is 256KB, which can be negligible. Also, the dwell time computation overhead of each block and the average dwell time for each group during GC should be considered. The dwell time calculation, involving a simple subtraction for a victim block during GC running in the background, has minimal computational overhead, ensuring no impact on the performance of serving write requests by the open block.

## V. Evaluations

In this section, we will present the experimental setup and compare the impact of extending the lifetime using our proposed method with multi-stream GC and state-of-the-art work. Furthermore, we will evaluate the performance and overheads of implementing our proposed method.

### A. Experimental Setup

The experiments are conducted on the SSDsim [29], which is an event-driven SSD simulator. Table I shows the detailed configuration of the evaluated SSD. There are two channels, and each channel consists of two chips. We use a four-plane architecture in one die [30]. Each plane consists of 512 blocks. A block has 1024 pages, so there are 524288 pages in a plane. The capacity of each page is 16KB. Thus, the total SSD capacity is 128GB. The ratio of over-provisioning (OP) space is set as 7%, which is the usual setup for enterprise SSD products. The value of Threshold$_{GC}$ is set to 10%. Table II summarizes the characteristics (e.g., read footprint, write footprint, duration, and the number of loops) of nine real-world workloads from MSR-Cambridge benchmark [31], which are a common testing benchmark for evaluating the flash-based storage systems [9], [11]. We implement extensions to support the recent TLC flash memory. The request size is multiplied by 4 times. Further, to trigger GC operations to wear the flash blocks, we repeat each workload with specific loops as shown in Table II, extending the SSD duration to three years, based on the minimum SSD lifetime [11]. The studies have shown that applications tend to have similar access behavior over days [2], [32]. We believe repeating workloads could simulate real long-time workloads. For each loop, we offset the start address of each request by its size to avoid the same address updating during replay. The evaluated comparisons are as follows:

- **Multi-Stream** combines the conventional greedy GC that selects a block with the highest number of invalid pages as a victim block [12], [13], with the multi-stream technique [14], where two streams for each plane: one for storing hot data and another for storing cold data. The identification of hot and cold data is based on the update frequency [27].
- **SmartNH** is the related work [9], aiming to enhance the lifetime of flash memory by strategically selecting worn-out blocks for heating to obtain the healing effect. To present a fair comparison, we excluded the hardware heating component, only comparing the strategy of data allocation that exploits the self-healing effect.
- **OSH** represents our proposed opportunistic self-healing method, where employ the same data hotness identification and multi-stream technique as the **Multi-Stream** method.

TABLE I
PARAMETERS FOR THE SIMULATED SSD.

| Parameters | Value | Parameters | Value |
|---|---|---|---|
| # of channel | 2 | # of page per block | 1024 |
| # of chip per channel | 2 | page capacity | 16KB |
| # of die per chip | 1 | SSD capacity | 128GB |
| # of plane per die | 4 | OP ratio | 7% [33] |
| # of block per plane | 512 | GC threshold | 10% [9] |

To assess the benefits of our proposal, two metrics are evaluated: 1) the cumulative distribution of dwell time to reveal the variation in dwell time among different blocks; and 2) the achievable P/E cycles to show the lifetime extension. Moreover, the overhead mainly comes from GC behaviors, which are evaluated from two aspects: 1) GC count; and 2) migrated valid page count. The thresholds used in **OSH** are $T_{blk}$ as 0.7, $T_c$ as 0.2, $T_i$ as 0.9, where the detailed parameter settings are discussed based on the sensitivity study in Section V.B.4).

TABLE II
THE CHARACTERISTIC OF THE EVALUATED WORKLOADS. (THE DURATION OF ALL WORKLOADS IS 3 YEARS)

| Workloads | Read Footprint(GB) | Write Footprint(GB) | Loops |
|---|---|---|---|
| hm0 | 0.86 | 9.94 | 2432 |
| rsrch0 | 0.14 | 12.27 | 1107 |
| stg0 | 5.64 | 6.27 | 1728 |
| wdev0 | 0.29 | 5.22 | 896 |
| prxy0 | 0.15 | 13.81 | 9788 |
| prn0 | 1.84 | 32.64 | 3932 |
| proj0 | 2.09 | 7.97 | 1544 |
| src20 | 0.64 | 14.33 | 1129 |
| usr0 | 2.23 | 7.56 | 1718 |

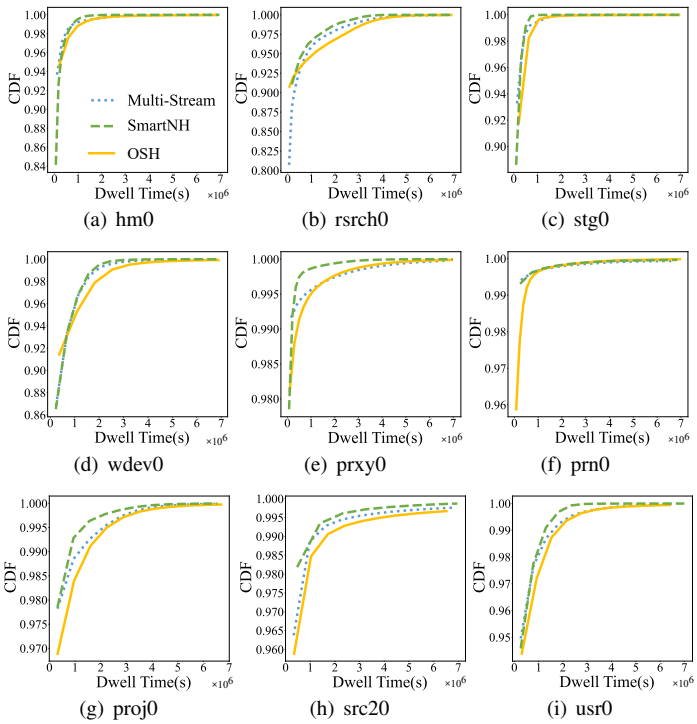



Fig. 7. The cumulative distribution of blocks' dwell time.

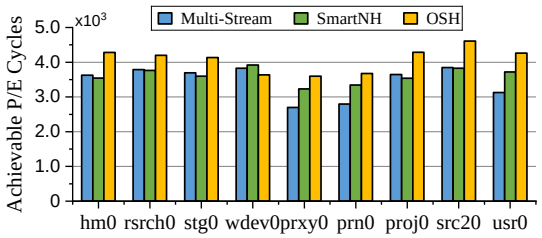



Fig. 8. The achievable P/E cycles (**OSH**: $T_{blk}$=0.7; $T_c$=0.2; $T_i$=0.9)

## B. Experimental Results

*1) The Cumulative Distribution of Dwell Time:* Fig. 7 shows the cumulative distribution of dwell time for blocks, comparing our proposed method (**OSH**) with the **Multi-Stream** and **SmartNH**. The x-axis represents the dwell time, and the y-axis is the probability of the dwell time. The results show that our proposed method creates varying degrees of uneven dwell-time distribution in the majority of workloads. The workloads, $\text{prxy}_0$, $\text{proj}_0$, $\text{src2}_0$, and $\text{usr}_0$, exhibit the significant imbalance in dwell time, as shown in Fig. 7(e), (g), (h) and (i), which potentially leads to a remarkable lifetime improvement. However, for the workload $\text{wdev}_0$, in Fig. 7(d), the whole distribution of the dwell time remains less uneven compared to that of other methods. Due to the low intensity of write requests in $\text{wdev}_0$, the GC process is not frequent on this workload, as shown in Fig. 9. As a result, the effect of self-healing from our method becomes not so effective. In conclusion, **OSH** achieves an uneven distribution of the dwell time in blocks, owing to the well-designed GC strategy, while it is less effective on workloads with low intensity of write requests. In the following, we will discuss the effects of each method on lifetime optimization in detail.

*2) The Extension of Flash Lifetime:* We use the achievable P/E cycles to evaluate the lifetime extension, as shown in Fig. 8. The demonstrated achievable P/E cycle counts for **Multi-Stream** and **OSH** have already subtracted the additional GC counts compared to **Multi-Stream**. From the results, **OSH** outperforms the **Multi-Stream** and **SmartNH** by achieving a 19.3% and 13.2% on average enhancement in flash lifetime. This is because **OSH** can achieve an uneven distribution among all blocks. Since the *GC Block Selection* of the **OSH** may not always choose the block with the maximum number of invalid pages, decreasing reclaim space may trigger more GC operations than methods using greedy GC, like **SmartNH** and **SmartNH**, **OSH** introduces the least additional GC counts for all nine workloads.

*3) Analysis of GC Behaviors:* We analyze the GC behaviors in detail in this subsection from the perspective of the GC count and the migrated valid page count.

**GC Count:** Although the GC count for **OSH** is marginally higher than other methods, as illustrated in Fig. 9, the increment is minimal. In the analysis of all nine workloads, the GC count for **OSH** is only about 2.3% higher than that for **Multi-Stream**

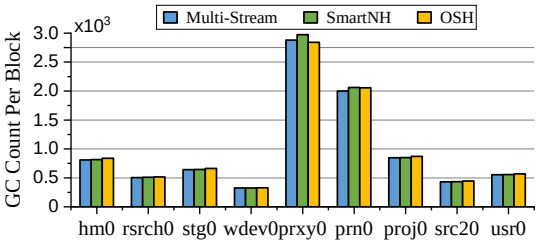



Fig. 9. The average GC count per block.

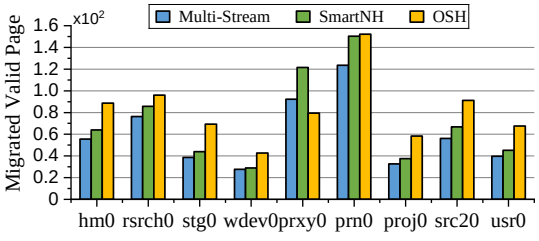



Fig. 10. The average migrated page count.

and 1.3% higher than for **SmartNH**. Therefore, the cost of the extra GC counts can be considered negligible.

**Migrated Valid Page Count:** To evaluate the additional GC impact because of the lower GC efficiency, we collect the average migrated valid page count during the GC process, as shown in Fig. 10. The **OSH** may not select the victim block with the most number of invalid pages, so we introduce the $T_i$ to control the effect of valid page migrations. By appropriately selecting $T_i$, the **OSH** exhibits an average increment of approximately 5% and 3% in the migrated valid page count compared to **Multi-Stream** and **SmartNH**, respectively. This increase is considered acceptable.

*4) Sensitivity Study:* The benefits of **OSH** are affected by the thresholds, $T_{blk}$ and $T_c$. **First**, Fig. 11 shows the achievable P/E cycles, the average GC count per block, and the average migrated valid page count of each GC when the value of $T_{blk}$ is fixed at 0.7 and vary the value of $T_c$ from 0.1 to 0.5 with a step size of 0.1. The reason for selecting these sets of $T_{blk}$ and $T_c$ values is based on Fig. 4. It can be observed that substantial improvements are achieved when $T_{blk}$ is approximately 0.7, with $T_c$ holding the same value. We also observe that by fixing $T_{blk}$'s value, as $T_c$ decreases, we can achieve a higher number of achievable P/E cycles. By applying these sets of values for $T_{blk}$ and $T_c$ in our proposal and testing them on nine real workloads. From the figure, the achievable P/E cycles increase as $T_c$ decreases (see Fig. 11(a)). However, this improvement is accompanied by an increase in the number of GC counts per block (see Fig. 11(b)), and in the number of valid page migrations during each GC process (see Fig. 11(c)). In conclusion, $T_c$ is set to 0.2 and $T_{blk}$ is set to 0.7 in this paper.

**Second**, when $T_c$ is 0.2 and $T_{blk}$ is 0.7, the value of $T_i$ is varied among 0.95, 0.9, and 0.85, as shown in Fig. 12. We observe that $T_i$ effectively limits the number of valid page migration during each GC process. From the figure, with the decrease of $T_i$, there is a slight increase in the number of achievable P/E cycles (see Fig. 12(a)), however, both the GC counts per block (see Fig. 12(b)) and the number of valid page migration per GC process (see Fig. 12(c)) also increase. Therefore, in this paper, we set $T_i$ to 0.9.

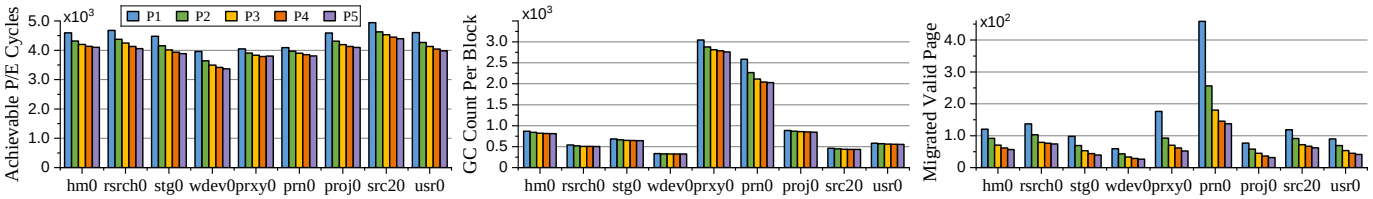
## VI. Related Work

To improve the flash lifetime, previous studies have proposed approaches from different aspects, including 1) utilizing the self-healing effect and 2) minimizing the number of valid data migrations during garbage collection and wear leveling.

**Self-Healing Effect.** Multiple prior works proposed to improve the flash lifetime through self-healing, which can dissipate electrons embedded in the oxide layer, thereby mitigating flash reliability degradation. There are two ways to achieve self-healing, either by hardware-assisted heating or extending the dwell time between two erase operations. On the one hand, high temperature can accelerate the self-healing procedure to improve flash lifetime. To assist the heat-accelerated healing method, Macronix's researchers redesigned a new flash architecture with heaters along each wordline, which can create the local high temperature in a flash chip. Wu et al. [5] added an extra die in a flash chip for internal heating operation. Lue et al. [6] designed a built-in heating plate on the top of wordlines, which provides extremely high temperature in about one second to efficiently postpone the lifetime of the target block in flash memory. Chen et al. [7] and Chang et al. [8] proposed a strategy to heat cells in batches to avoid influencing the performance of self-healing operations. Cui et al. [9] raised a heating method that dynamically adjusts the heating operation by tracking the dwell time between two P/E cycles and the write hotness of workloads. However, the above heat-accelerated healing schemes have several drawbacks. First, before performing heat operations, the valid data in the heating-target block should be migrated to a new block. This process is time-consuming, which is approximately one second for a block [6]. Second, frequent self-healing operations will be triggered during the middle or late lifetime of flash memory. This is because more blocks have low reliability and need to be recovered by self-healing. Third, high temperature also harms valid data, increasing their RBERs.

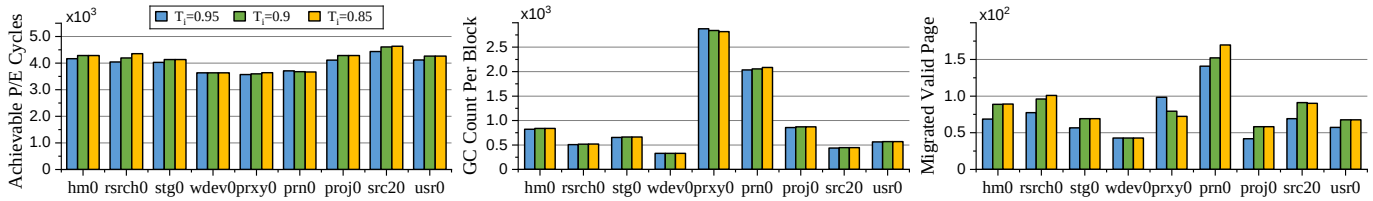
On the other hand, several works studied the impacts of dwell time on the achievable P/E cycles. Luo et al. [20] proposed to adjust the read voltage based on the recorded healing period. The model to expose the relationship between the stresses and self-healing on the flash cells was initiated by Mohan et al. [10], who exploited the healing effect by adding dwell time to extend the lifetime of SSDs. Moreover, a longer dwell time means more electrons can be dissipated [4], [5]. Unlike the above methods, our proposal focuses on extending the lifetime of 3D NAND flash memory via opportunistic self-healing.

**Garbage Collection and Wear Leveling.** Flash memory faces a notorious problem of a large number of valid data migrations during background GC and WL, which is brought by the out-of-update property of flash memory. Reducing the number of valid data migrations can ensure the flash lifetime, especially in a modern high-density flash with fewer P/E cycles. Prior GC works can be divided into two parts. The first is to use the GC victim block selection algorithms. For example, greedy GC is a widely adopted victim block selection algorithm




(a) The achievable P/E cycles. (b) The average GC count for per block. (c) The average migrated page count.

Fig. 11. The sensitivity study for $T_{blk}$ and $T_c$, where $T_{blk}$ is 0.7, only varying the $T_c$ values: 0.1 (P1), 0.2 (P2), 0.3 (P3), 0.4 (P4), 0.5 (P5).




(a) The achievable P/E cycles. (b) The average GC count for per block. (c) The average migrated page count.

Fig. 12. The sensitivity study for $T_i$ (0.95, 0.9, 0.85), and the value of $T_{blk}$ and $T_c$ are 0.7 and 0.2, respectively.

[34], which chooses a block with the highest number of invalid pages to erase. Further, Kwon et al. [35] proposed the FeGC, which improves the strategy of greedy GC by incorporating the invalidation history of pages in a block. Lin et al. [36] proposed to upgrade the GC strategy by using the past data update times. This approach increases both the efficiency and fairness of the GC process. The second is to allow data with the same characteristics to be grouped and written to different streams [13]–[16]. For example, Kang et al. [14] proposed a multi-stream SSD method, which maps application and system data with different lifetimes into SSD streams. Furthermore, Wang et al. [13] proposed to estimate the block invalidation time and group data with similar block invalidation time values into blocks. In addition, optimizing the overall performance of GC requires consideration in combination with the characteristics of the upper layer application, such as the real-time system [37]. Gao et al. [38] proposed to improve the latency and performance of SSDs by utilizing the variation of idle time to serve I/O requests among different chips without conflicts of internal data. The GC efficiency is closely related to the number of valid data migrations. To alleviate this issue, we use a combination of the block's dwell time and invalid page count to select a victim block. Different from the above works, we achieve self-healing by redesigning the GC algorithm to spontaneously control the dwell time of each block. Thus, we aim to improve our GC strategy while avoiding any negative impact on the achievable P/E cycle due to the low GC efficiency.

Unevenly wearing the memory cells [39] by the GC seriously impacts the lifetime. Thus the wear leveling can spread the erase and program operations across all the memory cells, rather than repeatedly targeting the same cells. Previous wear leveling strategies are implemented to make all blocks wear evenly. For example, Jimenez et al. [40] proposed to redesign the wear leveling algorithm by tracking the weakest cells and letting the strongest cells ease them. Yung et al. [41] proposed an effective block group-based wear leveling algorithm to separate the hot and cold data into different groups and relieve the I/O request in the granularity of a group. Chang et al. [42] proposed evenly distributing the healing cycles among all the flash. Different from the above works, our work considers the dwell time of blocks and strategically selects a Group for serving writes to make full use of all block's P/E cycles.

## VII. Conclusion

In this paper, an opportunistic self-healing method is proposed to enhance the lifetime of flash memory. To realize the uneven distribution of dwell time during the block's lifetime, three schemes are proposed and discussed. The *Block Partition Scheme* is designed to control the number of blocks between the *Active Group* and the *Healing Group*. The *GC Block Selection* is proposed to strategically select victim blocks from either group. The last one is the *Inter-Group Switching*, which periodically switches between blocks in the *Active Group* and blocks in the *Healing Group*, achieving uneven dwell-time distribution among blocks' lifetime. Evaluation results show that our proposal can extend the flash lifetime by 19.3% and 13.2% on average with near-free overhead, compared with the baseline and the related work.

## Acknowledgment

We sincerely thank anonymous reviewers for their constructive feedback. This work was supported in part by the Natural Science Foundation of Xiamen under Grant No. 3502Z20227023, the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11217020), the National Natural Science Foundation of China under Grant No. 62202396, and the China Fundamental Research Funds for the Central Universities under Grant No. 20720230072. The corresponding author is Qiao Li (liqiao@xmu.edu.cn).

## References

[1] D. Hong, M. Kim, G. Cho, D. Lee, and J. Kim, "GuardedErase: Extending SSD lifetimes by protecting weak wordlines," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*. Santa Clara, CA: USENIX Association, Feb. 2022, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/fast22/presentation/hong

[2] S. Jaffer, K. Mahdaviani, and B. Schroeder, "Improving the reliability of next generation {SSDs} using {WOM-v} codes," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 117–132.

[3] BocksFiles, "Western digital and toshiba talk up penta-level cell flash." 2019, https://blocksandfiles.com/2019/08/07/penta-level-cell-flash/.

[4] N. Mielke, H. P. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery effects in the distributed cycling of flash memories," in *2006 IEEE International Reliability Physics Symposium Proceedings*. IEEE, 2006, pp. 29–35.

[5] Q. Wu, G. Dong, and T. Zhang, "Exploiting heat-accelerated flash memory wear-out recovery to enable self-healing ssds," in *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, 2011.

[6] H.-T. Lue, P.-Y. Du, C.-P. Chen, W.-C. Chen, C.-C. Hsieh, Y.-H. Hsiao, Y.-H. Shih, and C.-Y. Lu, "Radically extending the cycling endurance of flash memory (to¿ 100m cycles) by using built-in thermal annealing to self-heal the stress-induced damage," in *2012 International Electron Devices Meeting*. IEEE, 2012, pp. 9–1.

[7] R. Chen, Y. Wang, D. Liu, Z. Shao, and S. Jiang, "Heating dispersal for self-healing nand flash memory," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 361–367, 2016.

[8] L.-P. Chang, S.-M. Huang, and K.-L. Chou, "Relieving self-healing ssds of heal storms," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–7.

[9] J. Cui, J. Liu, J. Huang, and L. T. Yang, "Smartheating: On the performance and lifetime improvement of self-healing ssds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 1, pp. 52–65, 2020.

[10] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan, "How i learned to stop worrying and love flash endurance," in *2nd Workshop on Hot Topics in Storage and File Systems (HotStorage 10)*, 2010.

[11] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime management of flash-based ssds using recovery-aware dynamic throttling." in *FAST*, vol. 3, 2012, pp. 1–6.

[12] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Performance Evaluation*, vol. 67, no. 11, pp. 1172–1186, 2010.

[13] Q. Wang, J. Li, P. P. Lee, T. Ouyang, C. Shi, and L. Huang, "Separating data via block invalidation time inference for write amplification reduction in {Log-Structured} storage," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 429–444.

[14] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed {Solid-State} drive," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

[15] J. Yang, R. Pandurangan, C. Choi, and V. Balakrishnan, "Autostream: Automatic stream management for multi-streamed ssds," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–11.

[16] T. Kim, D. Hong, S. S. Hahn, M. Chun, S. Lee, J. Hwang, J. Lee, and J. Kim, "Fully automatic stream management for {Multi-Streamed}{SSDs} using program contexts," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019, pp. 295–308.

[17] Y.-T. Chiu, "Forever flash," *IEEE Spectrum*, vol. 49, no. 12, pp. 11–12, 2012.

[18] D. Wei, L. Qiao, X. Chen, M. Hao, and X. Peng, "Srea: A self-recovery effect aware wear-leveling strategy for the reliability extension of nand flash memory," *Microelectronics Reliability*, vol. 100, p. 113433, 2019.

[19] K. Mizoguchi, K. Maeda, and K. Takeuchi, "Automatic data repair overwrite pulse for 3d-tlc nand flash memories with 38x data-retention lifetime extension," in *2019 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2019, pp. 1–5.

[20] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Heatwatch: Improving 3d nand flash memory device reliability by exploiting self-recovery and temperature awareness," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 504–517.

[21] B. S. Kim, J. Choi, and S. L. Min, "Design tradeoffs for {SSD} reliability," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019, pp. 281–294.

[22] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 94–101.

[23] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[24] YEESTOR, "9082hc ssd platform," 2018, https://www.yeestor.com/products/detail/i-14.html.

[25] Q. Li, M. Ye, T.-W. Kuo, and C. J. Xue, "How the common retention acceleration method of 3d nand flash memory goes wrong?" in *13th ACM Workshop on Hot Topics in Storage and File Systems*, pp. 1–7.

[26] S. Arrhenius, "Über die dissociationswärme und den einfluss der temperatur auf den dissociationsgrad der elektrolyte," *Zeitschrift für physikalische Chemie*, vol. 4, no. 1, pp. 96–116, 1889.

[27] K. Kremer and A. Brinkmann, "Fadac: A self-adapting data classifier for flash memory," in *Proceedings of the 12th ACM International Conference on Systems and Storage*, 2019, pp. 167–178.

[28] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed Solid-State drive," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*. USENIX Association, Jun. 2014.

[29] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 96–107.

[30] T. Pekny, L. Vu, J. Tsai, D. Srinivasan *et al.*, "A 1-Tb Density 4b/Cell 3D-NAND Flash on 176-Tier Technology with 4-Independent Planes for Read using CMOS-Under-the-Array," *IEEE International Solid-State Circuits Conference (ISSCC)*, 2022.

[31] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to ssds: analysis of tradeoffs," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 145–158.

[32] J. Li, Q. Wang, P. P. Lee, and C. Shi, "An in-depth analysis of cloud block storage workloads in large-scale production," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2020, pp. 37–47.

[33] S. Pang, Y. Deng, G. Zhang, Y. Zhou, Y. Huang, and X. Qin, "Psa-cache: A page-state-aware cache scheme for boosting 3d nand flash performance," *ACM Transactions on Storage*, vol. 19, no. 2, pp. 1–27, 2023.

[34] Z. Jiao, X. Zhang, H. Shin, J. Choi, and B. S. Kim, "The design and implementation of a Capacity-Variant storage system," in *22nd USENIX Conference on File and Storage Technologies (FAST 24)*. Santa Clara, CA: USENIX Association, Feb. 2024, pp. 159–176. [Online]. Available: https://www.usenix.org/conference/fast24/presentation/jiao

[35] O. Kwon, K. Koh, J. Lee, and H. Bahn, "Fegc: An efficient garbage collection scheme for flash memory based storage systems," *Journal of Systems and Software*, vol. 84, no. 9, pp. 1507–1523, 2011.

[36] M. Lin and Z. Yao, "Dynamic garbage collection scheme based on past update times for nand flash-based consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 61, no. 4, pp. 478–483, 2015.

[37] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 4, pp. 837–863, 2004.

[38] C. Gao, L. Shi, Y. Di, Q. Li, C. J. Xue, K. Wu, and E. Sha, "Exploiting chip idleness for minimizing garbage collection—induced chip access conflict on ssds," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, pp. 1–29, 2017.

[39] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, and T.-W. Kuo, "Garbage collection and wear leveling for flash memory: Past and future," in *2014 International Conference on Smart Computing*. IEEE, 2014, pp. 66–73.

[40] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: Improving nand flash lifetime by balancing page endurance," in *12th USENIX Conference on File and Storage Technologies (FAST 14)*, 2014, pp. 47–59.

[41] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee, "A group-based wear-leveling algorithm for large-capacity flash memory storage systems," in *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, 2007, pp. 160–164.

[42] Y.-M. Chang, Y.-H. Chang, J.-J. Chen, T.-W. Kuo, H.-P. Li, and H.-T. Lue, "On trading wear-leveling with heal-leveling," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.