Minimizing Retention Induced Refresh Through Exploiting Process Variation of Flash Memory

Yejia Di, Liang Shi, Congming Gao, Qiao Li, Chun Jason Xue and Kaijie Wu

Abstract—Refresh schemes have been the default approach in NAND flash memory to avoid data losses. The critical issue of the refresh schemes is that they introduce additional costs on lifetime and performance. Recent work proposed to minimize the refresh costs by using uniform refresh frequencies based on the number of program/erase (P/E) cycles. However, from our investigation, we find that the refresh costs still have a high burden on the lifetime performance. In this paper, a novel refresh minimization scheme is proposed by exploiting the process variation (PV) of flash memory. State-of-the-art flash memory always has significant PV, which introduces large variations on the retention time of flash blocks. In order to reduce the refresh costs, we first propose a new refresh frequency determination scheme by detecting the supported retention time of flash blocks. If the detected retention time is large, a low refresh frequency can be applied to minimize the refresh costs. Second, considering that the retention time requirements of data are varied with each others, we further propose a data hotness and refresh frequency matching scheme. The matching scheme is designed to allocate data to blocks with right higher supported retention time. Through simulation studies, the lifetime and performance are significantly improved compared with state-of-the-art refresh schemes.

Index Terms—Retention Time, Refresh Optimization, Process Variation, Flash Memory.

1 Introduction

NAND flash memory has been widely used in embedded systems, personal computers, and data centers due to the benefits of cost reduction from technology scaling and bit density improvement. Recently, the technology size has been scaled to approaching 10nm [2] [3], and the number of bits per cell has been advanced to 6 [4]. However, the technology scaling and bit density improvement bring in new challenges. One of the challenges is the degraded reliability issue [5] [6]. The direct impact on flash memory is the reduced number of supported program/erase (P/E) cycles. In this case, with the wearing of flash memory, some flash blocks become unreliable to store data for a satisfied long time, and data would be lost [7] [8]. The duration of time for which the data written in flash memory cells can be read reliably is called retention time. To avoid data loss, the straightforward scheme is to refresh the data [9], where data are copied to other free pages before the retention time is reached [10]. However, the key challenge for the refresh scheme is its high cost on the redundant read and rewrite operations. These costs will significantly impact the performance and lifetime of flash memory. Another challenge is the process variation (PV). PV has been identified as the general characteristic of state-of-the-art flash memory [11] [12] [13] [14] [15], which introduces significant endurance variations among flash blocks, representing that the supported retention time varied with flash blocks. In this work, PV on supported retention time of flash blocks is exploited to minimize refresh costs.

Refresh schemes can be classified into two groups: fixed-period refresh [16] [17] [18] [19] [20] and demand based refresh [21] [22] [23] [24]. For fixed-period refresh schemes, they are proposed to refresh data with a fixed frequency. In a fixed period, all blocks are refreshed to promise the integrity of data. For demand based refresh schemes, they are proposed to refresh data only when there are approaching errors in the data. The key drawback of the demand based scheme is that many refresh operations would be burst, which is burdensome for real devices. In this work, the fixed-period scheme is applied for flash memory by default. However, refresh as redundant operation has bad effects on the lifetime and performance. There are many works on reducing refresh operations [10] [21] [25], which can be classified into three groups: reprogramming in place [10], increasing the reliability of ECC [21], and grouping hot and cold data storing in the same block [25]. However, all these work use the number of P/E cycles to determine the refresh frequency, where many refresh operations are unnecessary considering PV. PV has been identified as a common feature of transistors, which presents significant endurance variations and large lifetime potentials among flash blocks [11] [12] [13] [14] [15]. Many works have been proposed to exploit the variations to improve the performance and lifetime. For example, Jimenez *et al.* [11] proposed to improve the lifetime by relieving the stress on weak-endurance flash pages. Pan *et al.* [12] and Yang *et al.* [26] exploited bit error rate (BER) as the metric of varied endurance, and proposed BER-aware wear leveling to lengthen the lifetime. Shi *et al.* [15] exploited various write speeds for flash blocks to improve performance considering the PV. However, none

- *Y. Di, L. Shi, and C. Gao are with the College of Computer Science, Chongqing University, Chongqing, China. E-mail: {yejia.di.cqu, shi.liang.hk, albertgaocm}@gmail.com.*
- *Q. Li and C. J. Xue are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: qiaoli045@gmail.com, jasonxue@cityu.edu.hk.*
- *K. Wu is with Tandon school of engineering, New York university, New York, USA. E-mail: kaijie@gmail.com. Corresponding author: Liang Shi*

An earlier version of this work has been accepted in the Proceedings of the 19th Design, Automation & Test in Europe (DATE), 2016 [1].

of these works exploit the PV characteristics to relieve the refresh costs.

In this paper, a scheme is proposed to minimize refresh costs through exploiting the PV characteristics of flash memory. There are two findings which motivate the proposed scheme. First, with PV, the real endurance can sustain lower refresh frequency than the one determined by P/E cycles. Second, update intervals of data are various, some long, some short. As refresh only occurs when the update interval of data is larger than the supported retention time reciprocal to refresh frequency of flash block, there will be a matching allocation for data and block with minimized refresh operations. According to the motivation, **we propose the first work on exploiting PV for reducing refresh operations with two methods**. First, a refresh frequency determination method is proposed. The method is designed through detecting the supported retention time of flash blocks, which is highly correlated to the refresh frequency. With the detected refresh frequency, a multiple-rank refresh queue is designed where blocks are maintained in the related refresh rank for fixed-period refresh. Second, a data and block matching method is proposed to allocate data into blocks with right higher supported retention time. As the supported retention time of flash blocks have been detected, the update intervals of data are detected first. And then three regions are divided for storing different kind of data, cold region for cold data, warm region for once-updated data, hot region for hot data. Based on the goal of the proposed work, the comparison between update interval and supported retention time instead of the specific update interval is gotten when refreshing, which is more valuable and straightforward. And then the data are allocated to matching regions for refresh minimization. Finally, implementation is realized according to the above methods. The proposed approaches are evaluated with a widely used SSD simulator [27] and several popular workloads [28]. Experimental results show that the performance and lifetime of flash memory will be significantly improved with the proposed work. This paper is an extended version of our previous work [1] with several new contributions. The major contributions of this work are as follows:

- A refresh frequency determination scheme is presented, where refresh frequencies of flash blocks are exploited;
- An online data and block matching scheme is presented, where blocks are allocated for related data;
- A new design on garbage collection is proposed, where different GC policies are applied for related regions;
- A new design on wear leveling is proposed taking advantage of the refresh queue;
- Other factors, such as temperature, are discussed;
- A set of overhead analysis and experiments are presented to show the advantages of the work.

The rest of the paper is organized as follows. In the next section, the background and related works are presented. Section 3 presents the problem of refresh schemes and motivation. In Section 4, the proposed scheme is presented in detail. Section 5 discusses the experimental results and analysis. We conclude the paper in Section 6.

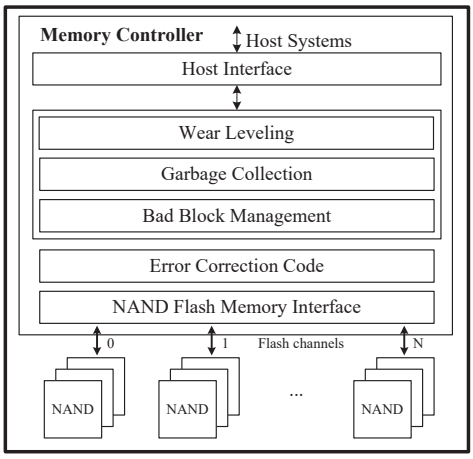


Fig. 1. The architecture of flash based storage devices.

2 Background and Related Work

2.1 Background

2.1.1 SSD Architecture

Flash based storage devices are generally composed with several flash chips. And flash controller is configured to manage these flash chips based on the characteristics of flash memory. The characteristics of flash memory are as follows: On the one hand, the flash memory must be erased before programming. This is because the erase operation unit is a flash block, while the programming operation unit is a flash page. Due to the asymmetric operation units, out-of-place update is required, which is realized by invalidating the original data in the flash page first, and then writing the updated data to free pages. On the other hand, with increasing erases, the flash memory may be worn out. That means that the lifetime of flash memory is limited, where each flash block can only be erased with limited P/E cycles [29] [30].

There are many functional components required in the memory controller [31], as shown in Figure 1. Due to the out-of-place updating scheme, the available space will be reduced with data updates. Once the available storage space falls below a given threshold, garbage collection (GC) will be triggered to reclaim the invalidated pages in the flash memory [32]. The operation of GC is to find a block, and erase it making P/E cycles increased. However, flash block will be worn out after erased limited P/E cycles. Once an amount of flash blocks are worn out, the flash memory will not be used any more. Bad block management (BBM) is designed to record flash blocks which are worn out. Wear leveling (WL) is designed to level the wearing among flash blocks for maximized lifetime [26] [12] [33]. Traditional WL mechanisms use P/E cycles as the wearing metric, where two schemes are widely studied, dynamic WL and static WL. Dynamic WL allocates blocks with the least P/E cycles for new data first. While the static WL is designed to move cold data from the block with the low number of P/E cycles to that with high number of P/E cycles. The foundational cause of worn-out flash block is that the error number is out of the capability of error correction code (ECC). ECC is designed to correct errors for original data when reading with redundant codes called parity codes. Before data is programmed, a parity code is produced by encoding the

TABLE 1
The comparison among prior related works.

Works	Principle	Metric for ①	Method for ②	Pros	Cons
Cai *et al.* [10]	①	P/E cycles	–	reducing some unnecessary refresh	unnecessary refresh due to PV
Park *et al.* [21]	①②	P/E cycles	dynamic vertical stripping	improving error correction redundancy for cold data	uncustomed hot and cold separation; still needing refresh mechanism
Luo *et al.* [25]	①②	P/E cycles	data separation	skipping refresh for hot-data blocks	not considering the block retention
Di *et al.* [1]	①②	BER	block allocation	considering PV; allocating related blocks to data	burst refreshes due to demand based refresh mode; offline data hotness identification

flash-page data, and then programmed to the out-of-band of flash page with the flash-page data. During read operations, the parity code and the data are read out, and then decoding operation is conducted with errors corrected.

2.1.2 Reliability Issue during Retention Time

Retention time errors have been identified as the dominant errors of NAND flash memory [8]. Many works have been proposed to study the characteristics [8] [34]. In conclusion, there are two significant characteristics of retention time errors as follows.

Retention time is limited and decreased with wearing. NAND flash memory is non-volatile, owning to the surrounding dielectrics called oxide. However, the oxide can be destroyed by the high program and erase voltages, resulting in charges trapped [35]. In this case, programmed charges can be leaked away gradually [36]. Figure 2 shows an example on the charges leakage for a 2-bit per cell flash memory. After a period of time, the charges stored in the cell can be leaked, making the cell states approaching to a lower cell state. When the voltage is shifted to a lower state, the data would be read incorrectly. Thus, the data can only be stored in flash memory with a limited retention time. Besides, with the increasing of wearing of flash memory, the charge leakage will also be increased with trapped charges. Therefore, the limited retention time of flash block will be decreased with wearing.

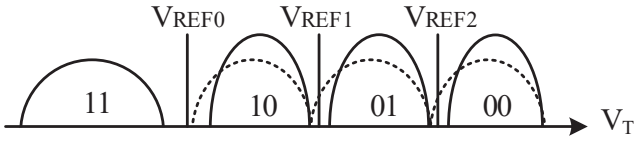


Fig. 2. **The threshold voltage distribution of 2-bit per cell flash memory.** Where the voltage distribution of solid lines and dotted lines represent the original states and the shifted states after a long retention time, respectively.

Errors are proportional to the data retention time. The leakage of charges is a gradual process. The longer the retention time is, the more the leaked charges are [37] [35]. Thus, if the data have been stored in flash memory over a long period of time, the retention errors will be higher.

2.1.3 Refresh Schemes

Refresh is a highly recommended scheme for alleviating retention errors. Before overwhelming error occurred, a rewrite operation is performed for another long retention time [9] [8] [10] [21] [25] [38] [39] [40]. There are two types of refresh schemes widely studied in previous works, fixed-period refresh [16] [17] [18] [19] and demand based refresh [21] [22] [23] [24]. Figure 3 shows the operation manners of the two types of refresh schemes. Assume that the refresh period is T_R which is reciprocal to refresh frequency, and refresh unit is N_B.

- Fixed-period refresh has been equipped as the default refresh scheme for DRAM. As shown in Figure 3, the refresh period is averaged to each bank, banks will be refreshed one by one every a small interval, $\frac{T_R}{N_B}$. Until the refresh period approaches T_R, all banks are refreshed.
- Demand based refresh is triggered when the storage time is up-to the refresh period T_R.

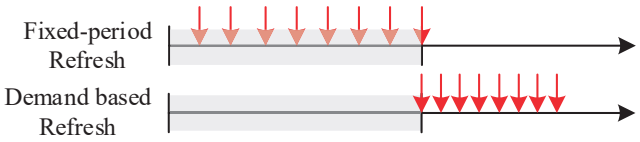


Fig. 3. **The refresh schemes.** The gray field represents the refresh interval, and the red arrow represents the refresh operation.

With different refresh schemes, the refresh efficiencies are also different. For demand based refresh schemes, all flash blocks are only refreshed when they have stored data for T_R. However, if many blocks are programmed at the same time, the refresh operations on these blocks will also be triggered at the same time. In this case, burst refresh operations will delay host requests with large refresh latencies. For the fixed-period refresh scheme, it also can be applied for flash memory. Like DRAM, the fixed-period refresh scheme is recommended for flash memory.

2.2 Related Work

2.2.1 Refresh Reduction Schemes

The key issue for the refresh schemes is that it would bring in additional operations, which has significant impact on performance and lifetime. There are two principles on refresh reduction: ①reducing unnecessary refresh operations; ②skip refresh operations with all data invalidated. There are many related works on refresh reduction, as shown in Table 1. The prior three works are related state-of-the-art works. However, none of these works considered the existence of PV. Compared with the conference version, Di *et al.* [1], all approaches are extended to make the proposed work complete and friendly. The contribution is presented in Section 1. In this work, the scheme is aimed at reducing refresh operations. Besides, there are also works on reducing retention errors to avoid refresh operations, such as in-place programming [10] or accurate read voltages [8]. In addition, refresh costs can be individually reduced through improving refresh efficiency, such as compressing refresh operations [41]. All these works are orthogonal to the proposed scheme.

For DRAM, the retention time of DRAM cells are also non-uniform and multi-rate refresh techniques have been proposed to reduce the DRAM refresh overheads [17] [19]

[42]. In [17], dummy refresh operations are set for skipping refresh operations for reliable rows without hardware overhead. Liu *et al.* [19] proposed to group rows into bins depending on their measured refresh rate with bloom filters, and apply different refresh rates to bins. Besides, retention time of some cells can change at runtime due to variable retention time (VRT), thus a VRT-aware multi-rate refresh scheme [42] is proposed. However, these methods cannot be applied on flash memory directly. This is because that the refresh frequencies of rows of DRAM change slightly while the refresh frequencies of flash blocks are decreased with the wearing. In this paper, a method on detecting refresh frequencies of flash blocks under PV is proposed.

2.2.2 Process Variation

PV is a natural characteristic of semiconductors, which introduces significant reliability variations on many kinds of memories, such as DRAM [42], Spin Transfer Torque Magnetic RAM (STT-MRAM) [43], flash memory [13] and so on. Here, we consider the PV of flash memory. Recently, many works are proposed on identifying the PV characteristics. Woo *et al.* [13] proposed a new wear index, which took several properties into account including erase counts and program latency, forming a linear model for measuring the wearing degree of flash blocks. Measured results show that the P/E cycles of the failed blocks are far beyond the guaranteed P/E cycles provided by the manufactories, and are not affected by the locations of the flash blocks. Besides, the actual values of P/E cycles differ from flash blocks to flash blocks. Pan *et al.* [12] presented a dynamic BER-based greedy wear-leveling algorithm that used BER statistics as the wear-out pace for flash blocks, and guided dynamic data swapping among flash blocks to fully maximize the efficiency. Jimenez *et al.* [11] observed that the BERs of pages in one block are significantly varied. This motivates them to reduce the stress on the weakest pages for endurance enhancement. What's more, Meza and Mutlu *et al.* [14] presented the first large-scale study of flash-based storage reliability in this field. However, there are no works exploiting the retention time variations to minimize refresh cost.

2.2.3 Hot and Cold Data Separation

Several works are proposed on separating hot and cold data. Park *et al.* [44] proposed a novel hot data identification scheme adopting multiple bloom filters to efficiently capture finer-grained recency as well as frequency. In this way, the overhead is reduced while the performance is improved. In [45], three hot/cold data separation policies are evaluated, including 2-level LRU, multiple bloom filter, and dynamic data clustering. The evaluation results show that dynamic data clustering performs the best on performance. However, these hot and cold data separation methods need several memory overhead. Actually the separation can be conducted not acutely for some goals. Chang *et al.* [46] observed that small size data are usually updated frequently and proposed to separate data by size. Luo *et al.* [25] found that a very small fraction of data pages is the destination of nearly 100% of the write requests, and then proposed to separate data by the time of the last writes. Differently, in this work, the hot and cold data separation method is proposed by comparing with the supported retention time of flash blocks with two advantages. First, PV is exploited for refresh minimization, where the supported retention time of flash blocks are exploited. Comparing with the supported retention time of flash blocks, the update interval of data can be reflected directly. Second, the goal of the paper is to minimize refresh, the comparison between the supported retention time of flash block and update interval of data is more valuable. Because that the refresh can be minimized if all the data are stored in the blocks with a larger supported retention time, which is guided by retention.

3 Problem Statement and Motivation

3.1 Problem of Refresh Schemes

With technology scaling and density improvement, the supported retention time of flash memory is decreased. To prolong the retention time, refresh is the straightforward method. Besides, with the wearing of flash memory, the refresh period is decreased. In detail, Cai *et al.* [10] measured the required refresh periods under different lifetime stages when the ECC can correct 512 error bits. The measurement results show that the refresh period is 3 years when the P/E cycles is around ∼3k, while decreased to only 3 days when the P/E cycles is around ∼150k. In other words, refresh is an efficient method for lifetime improvement.

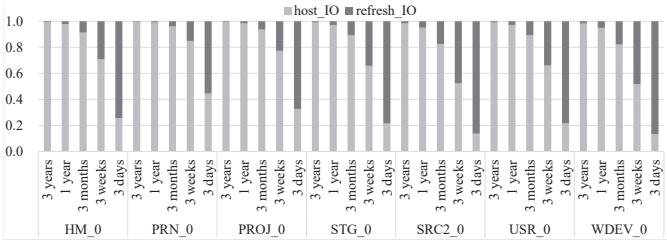
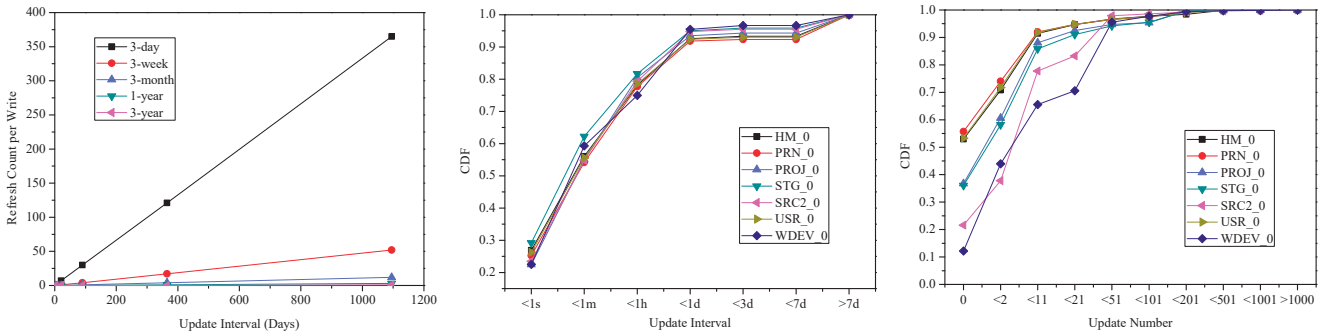


Fig. 4. The percentage distributions of numbers of host I/Os and refresh I/Os under the five refresh periods.

However, refresh operation accompanied with read and rewrite operations is costly for flash memory, especially for state-of-the-art low reliability devices. The lower the refresh period is, the larger the impact on flash memory will be. To reduce the refresh impacts, adaptive method [10] is proposed to apply different refresh period for each lifetime stage. With the wearing of flash memory, the refresh period is decreased for reliability. As measured in [10], five refresh periods, 3 years, 1 year, 3 months, 3 weeks and 3 days, are configured for the corresponding lifetime stages. With the above optimizing methods, we measured the refresh effects. Figure 4 shows the percentage distributions of executed I/Os including host I/Os and refresh I/Os in I/Os. The measurement is conducted on SSDsim simulator [27] under several widely used traces, which are introduced in the experiment section. As shown in Figure 4, with the wearing of flash memory, the number of refresh I/Os becomes larger and larger. Clearly, at the young stage of flash memory, when the refresh period is high, the most executed I/Os are host I/Os. While at the old stage of flash memory, low refresh period is applied, the most executed I/Os are occupied by the refresh I/Os.

With large percentage of refresh I/Os, the most improved lifetime will be consumed by refresh I/Os. Besides, refresh induced read and rewrite operations may conflict



(a) (b) (c)

Fig. 5. **The update interval effects and characteristics on refresh overhead.** (a) Refresh counts vary with the update interval of data. (b) The update intervals of requests are different. (c) The update numbers of data are different.

with host I/Os, which will have impacts on the performance. Thus, it is urgent to reduce the number of refresh I/Os, especially at the end lifetime of flash memory.

3.2 Motivation

Though refresh can prolong the lifetime of flash memory, the improved lifetime is mostly spent by the refresh operations. Thus, it is necessary to minimize the refresh operations. Here, two phenomena give us motivations, based on which the proposed work is designed.

3.2.1 PV Effects on Refresh

In the state-of-the-art adaptive-rate FCR method, P/E cycles are used as the endurance metric to progressively increase refresh frequency, so that unnecessary refresh can be minimized. However, with PV, the endurance among blocks is significantly varied under some P/E cycles.

PV is a critical characteristic of the state-of-the-art flash memory, especially for the small size and high density one. PV is introduced by the different physical cell margins and oxide thickness, resulting in different endurance of flash blocks. Pan *et al.* [12] presented that the P/E cycles of all the flash blocks fall into the range of [15000, 24600], all larger than the presented one given by industry. Woo *et al.* [13] also measured the P/E cycles of blocks. They presented that the lowest-endurance blocks can sustain 4999 cycles, and the average cycles are 8524 with the standard deviation of 1318. All the measured results show the following observations: First, the highest-endurance blocks can sustain ten-time thousand P/E cycles. Second, the P/E cycles of the blocks with the lowest endurance are only thousands still larger than the one given by industry. Thus, the real endurance among blocks is different and higher than the predefined endurance from industry. In this work, the real endurance that is the supported retention time under PV will be exploited to minimize unnecessary refresh operations.

3.2.2 Update Interval Impacts on Refresh

The update interval of data also has large effects on refresh operations. If the update intervals are smaller than the supported retention time of the flash block, that is all flash pages have been invalidated before refresh, the refresh operation can be skipped. As shown in Figure 5(a), with increasing update interval, the refresh count per write increases. While the refresh count varies with refresh frequency. The larger the refresh frequency is, like 3-day refresh, the larger the refresh count will be. Thus, if the data with large update intervals are allocated to blocks with low refresh frequency, the refresh count can be minimized.

Further, we measured the update characteristics of data as shown in Figures 5(b) and 5(c). Figure 5(b) shows the update intervals of requests, where '$< 1s$' in x-axis represents that the update interval is smaller than 1 second, and so on for 1 minute, 1 hour, 1 day, 3 days, 7 days and longer. Almost 90% of request updates are triggered within one day, while the remaining data will be updated after 7 days or longer. However, these requests may be from the same data. Besides, Figure 5(c) shows the update numbers of data, where '*0*' in the x-axis represents there are no update for the data, while '< 2' means that the data are updated once, and so on for 10, 20, 50, 100, 200, 500, 1000 times and more times. From the figure, we can find that most of data are only updated within ten times. Combining the observations from the two figures, we can find that some data will be frequently updated with small update interval; While some will not be updated with a large retention time. Considering the differences among update intervals of data, refresh can be minimized with data allocation.

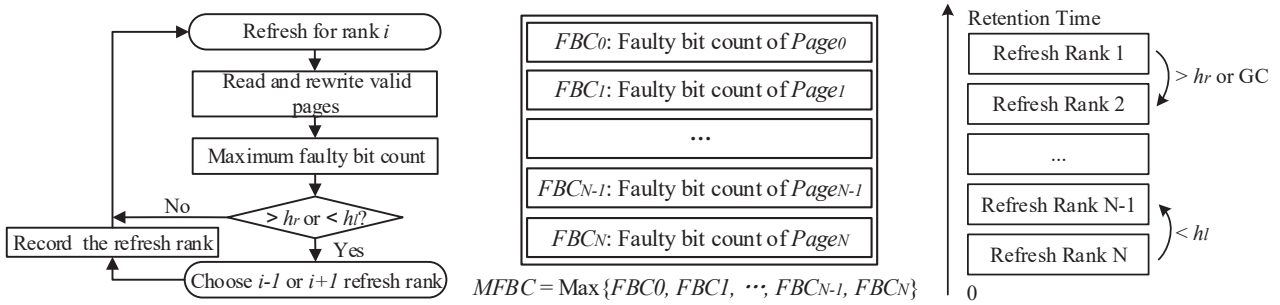
Taking all the above analysis into the consideration, if the data especially infrequent-updated data are allocated to block with larger supported retention time, the refresh operations can be minimized.

4 Process Variation Exploration Scheme on Retention Induced Refresh Minimization

4.1 Overview

In this paper, a process variation exploration scheme on retention induced refresh minimization is proposed to minimize refresh operation. With this scheme, both the performance and lifetime can be improved. There are two key ideas for the scheme. The first one is to exploit real refresh frequencies for flash blocks under PV. Then unnecessary refresh operations can be minimized. The second one is to allocate data to blocks with a lower refresh frequency. Then, the refresh operations can be skipped with invalidated data. Two step-by-step methods consist of the whole scheme. First, refresh frequency determination scheme is proposed to exploit the real refresh frequencies of flash blocks (Section 4.2). Second, data and block matching scheme is proposed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2858771, IEEE Transactions on Computers



(a) The detailed procedure. (b) Maximum faulty bit count determination. (c) Available refresh rank chosen method.

Fig. 6. **The overall procedure of retention time detection scheme.** Where the maximum faulty bit count (MFBC) is determined in figure (b). And the refresh rank is chosen according to figure (c), and the black curved arrow represents that a shorter retention time is chosen.

to allocate data to blocks with matching refresh frequencies for minimized refresh operations. In this method, both the management on wear leveling and garbage collection are presented (Section 4.3). Besides, the implementation of the proposed scheme is presented and the overhead is also analyzed (Section 4.4). Notably, the proposed scheme is aimed at MLC NAND flash memory. This work can be easily extended to 3D NAND flash memory, as there are larger endurance variations [47].

4.2 Refresh Frequency Determination

In this section, PV is first exploited by detect the retention time of flash blocks called retention time detection method. Since retention time is highly correlated to the refresh frequency, various refresh frequencies can be then determined.

4.2.1 Retention Time Detection

Figure 6 presents the overall procedure of the retention time detection scheme. The detailed procedure is shown in Figure 6(a). Refresh operation is executed for each flash block according to its predefined retention time which is measured in Figure 6(c). During refresh, valid pages in the flash block are read one-by-one. Then, these valid pages are written to other free pages. Over the operations, the maximum faulty bit count is obtained which is derived based on the process presented in Figure 6(b). Two thresholds h_r and h_l are set for the maximum faulty bit count, where h_r is larger than h_l. If the maximum faulty bit count is higher than h_r, the refresh operation of the flash block must be executed with a lower retention time called the next refresh rank for reliability. Otherwise, if the maximum faulty bit count is lower than h_l, the supported retention time of the flash block is regarded longer than the current one, and the prior refresh rank is selected for the flash block. Otherwise, the flash will be refreshed with the current rank.

In detail, the maximum faulty bit count is determined as shown in Figure 6(b), where the maximum faulty bit counts of valid pages are gotten. The maximum faulty bit count is used to judge whether the flash block can sustain the retention time or not. If the faulty bit count is lower than h_l, the flash block is strong enough to sustain larger retention time. If it is higher than h_r, the flash block becomes too weak to sustain the retention time. Otherwise, the flash block can continuously sustain the retention time. Here, the maximum faulty bit count is compared with thresholds h_r and h_l, where h_r is set to be lower than the error correction capacity of ECC, and h_l is small almost equal to 0. *The threshold setting is also based on a conservative purpose and the reason will be described in the foundation of the scheme.*

Besides, the available refresh rank chosen method is presented in Figure 6(c), where each refresh rank represents a retention time. And the refresh rank with the longest retention time which can be supported by flash block is chosen as its refresh rank, so that the purpose of reducing refresh operations can be achieved. In the approach, there are several refresh ranks (1,...,N) available with significant retention time differences. Among these refresh ranks, refresh rank 1 represents the longest retention time can be supported, while refresh rank N represents only the shortest retention time can be supported. In the beginning, the supported retention time of all flash blocks are regarded as the longest one. When the block is refreshed, the retention time will be detected by comparing the maximum faulty bit number of the flash block. If the maximum faulty bit number is higher than the threshold h_r or lower than the threshold h_l, the refresh rank of the flash block will become the next one or the prior one as the curved black arrow shown in Figure 6(c). Note that if the current refresh rank is 1, its prior refresh rank is also rank 1. While if the current refresh rank is N, and its next rank should be chosen for the flash block, the flash block is considered as a bad block. Notably, larger N means finer grained retention time separation, which will bring in larger benefits. Besides, when the block is reclaimed by garbage collection (GC) before refresh operation, the refresh rank of the block is also regarded as the next one. There are two reasons for the refresh rank changing. The first one is that GC will make flash blocks wearing, thus changing refresh rank is reliable to them. The second one is that only refresh is triggered, refresh overhead is introduced, but the refresh rank of the block will be rejudged again for later refresh according to the maximum faulty bit count.

The Foundation: The foundation of the proposed method is from two aspects. First, the faulty bit number of flash memory behaves an increasing feature. The faulty bits are mainly resulted from wearing errors and retention errors [7] [10]. Wearing introduces hard errors, bringing permanent harm to flash and increasing with erase operations, where the supported retention time as one representation of endurance will also be decreased. Retention errors are soft errors, which will be removed after erase but are dominant errors under fixed erase cycles. The longer the retention time is, the more the faulty bit number will be. Thus, with the wearing of flash memory, the supported retention time should be decreased, and it can be reflected by the faulty bit

number after a period of retention time. Second, ECC is configured for error correction, but the error correction capacity is limited. Thus, the faulty bit number must be too large to be corrected by ECC. If faulty number is high, it represents that the retention time cannot be supported continuously. Otherwise, longer retention time can be supported. However, errors are random occurred. For conservation, a threshold h_r which is less than the error correction capacity of ECC is set for the proposed method. And a small h_l is set taking some random errors into consideration. Based on the two aspects, if the faulty bit number is large, the flash block is regarded that it cannot support the retention time any more. Otherwise, it is strong enough to support longer retention time.

4.2.2 Refresh with Determined Frequency

Once the supported retention time of flash blocks are detected, the highly correlated refresh frequency can be determined. For fixed-period refresh scheme with a constant retention time of each refresh rank, the key issue is to get the number of flash blocks in each rank. In this way, the refresh frequency of each refresh rank can be determined which is reciprocal to the refresh interval determined by $\frac{T_R}{N_B}$ in the Background Section. Here, a multiple-rank queue is designed where blocks with the supported retention time are maintained in each refresh rank. With the number of blocks in each refresh rank, each refresh rank maintains a corresponding refresh frequency, and refresh operations are triggered for blocks one by one for each refresh interval.

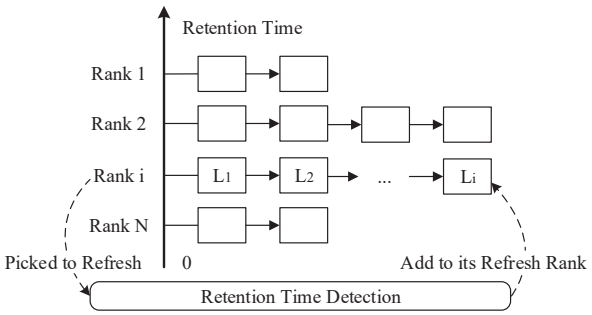


Fig. 7. **A multiple-rank queue is designed for refresh with various frequencies.** Li represents the location information of flash block.

There are two issues which should be solved in the multiple-rank queue design. The first one is that the refresh location of the block must be given for accurate refresh. The second one is that the supported retention time of flash block will be dynamically adjusted. In this way, the number of flash blocks and the refresh frequency will both be changed for the related refresh ranks. To solve these two issues, the multiple-rank queue is designed as follows. As shown in Figure 7, the multiple queue is maintained with N refresh queues as the refresh rank number is N, where each rank is maintained with a refresh queue. After retention time detection, each flash block is added to the refresh queue maintained by the detected refresh rank. First, the location information of the blocks are maintained in the multiple queue for accurate refresh. For each refresh queue, the refresh is executed on flash blocks from head to tail in the refresh queue. Each time, the location of the to-be refreshed block is gotten, and then read-and-write operations are performed for pages of the block. As described above, each refresh rank manages its refresh-interval refresh operations. Second, with refresh operation, the supported retention time of the flash block will be detected as described in the above section. If the supported retention time of the flash block are changed, the flash block numbers of prior and new detected refresh rank are both changed. The location information of the flash block should also be migrated from the prior detected refresh rank to the head of queue at the new detected refresh rank. The reason for adding it to the head of the refresh queue is that refresh can only be triggered for the flash block at the next refresh round, in this way it will be refreshed in priority for data integrity.

4.2.3 Temperature Discussion

Temperature may also affect the retention time of flash cells. This is because the leakage increases with temperature thereby shortening the retention time. Study has been conducted to investigate this issue and reducing refresh frequency when temperature falls has been suggested [38]. While the optimization goal is different from this work, the conclusion and suggestion of the reference study are consistent with this work, since our scheme also reduces refresh frequency. It can be seen that if all other conditions remain the same, applying the proposed technique will only result in a lower temperature, and hence results in an even larger increase in supported retention time. We also experiment the proposed technique on two different temperatures to show its effectiveness, while leave the mathematical modeling to future work due to page limit.

4.3 Data and Block Matching

In this section, a data and block matching method is presented by allocating data to blocks for refresh minimization. The basic idea is to allocate data into blocks with right higher supported retention time. Here the required retention time of data is called as update interval. First, the update intervals of data are detected according to the detected supported retention time of flash blocks. Second, a matching method is proposed according to the update interval of data and the supported retention time of flash blocks.

4.3.1 Update Interval Detection

According to the update interval effects on refresh in Section 3.2 as shown in Figures 5(b) and 5(c), we find that some data will be frequently updated with small update interval; while some will not be updated with a large retention time. In addition, the update intervals of frequent-updated data are high-probably lower than one day. In this method, the recent update interval is used as the update interval of data. What's more, the recent update interval can be reflected by the refresh operation as the supported retention time of flash blocks have been detected. If data are updated before refresh, the update interval of the data is regarded smaller than the supported retention time of the flash block. Otherwise, the update interval of data is regarded larger than the supported retention time of the flash block. In this way, the relationship between update interval of the data and the supported retention time of flash blocks is acquired.

As the goal of the method is to minimize refresh through allocating data into blocks with a larger supported retention

time, the retention time relationship between flash block supported and data required becomes more valuable.

4.3.2 Matching Approach

Taking advantage of the relationship between the update interval of data and the supported retention time of data, the matching approach can be designed easily according to the basic idea of the proposed work. Figure 8 illustrates the overall procedure of the matching approach. As shown in the figure, the location information of blocks are maintained by the multiple-rank refresh queue. According to the ranks, these blocks are grouped into three regions, cold region, warm region, and hot region. Cold region is designed for storing data which are updated infrequently, while hot for frequently, so that the supported retention time of allocated block is higher than that of update interval and refresh can be minimized. The processes ① to ⑥ are designed to make the data and block matched. Besides, warm region is designed to filter data updated once as shown in Figure 5(c). In the following, these processes will be described in detail.

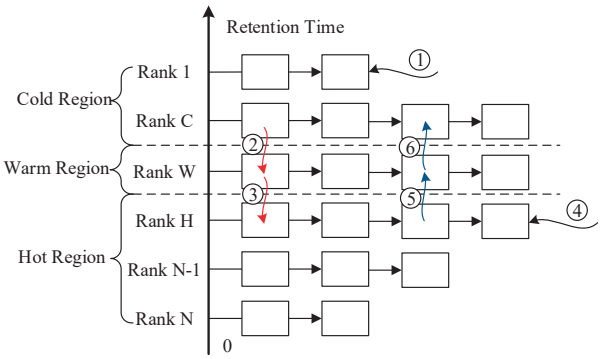


Fig. 8. The overall procedure of the matching approach.

Initially, all data are stored in the cold region as infrequent-updated data (①). If the data are updated before refresh operation, the data are regraded with a smaller update interval than the supported retention time of the block as described above. Then, the data are updated into warm region (②). Otherwise, the refreshed data will be stored in cold region continuously (①) as they would cause more refresh operations if allocated to other regions. All new data will first allocated to the cold region (①). There are two reasons for the design. The first one is that blocks in cold region can support longer retention time which means that these blocks can sustain larger P/E cycles, so that the wearing of writing frequent-updated data once is negligible. The second one is that frequent-updated data will be updated in a small period. Thus frequent-updated data will be moved into hot region in time and the invalidated space in cold region can be recycled for the following infrequent-updated data. While for the infrequent-updated data, directly allocating them to cold region is the best choice.

In the warm region, if the data are updated before refresh operation, the data will be updated into hot region (③) as the frequent-updated data. When refreshed, the valid data are rewritten into the cold region for longer retention time (⑥). Besides, if there are not enough space in cold region for infrequent-updated data or new data, the warm region will be allocated for the data.

All the frequent-updated data are updated into the hot region where flash blocks support smaller retention time. While as these data will be updated frequently less than 1 day, the supported retention time is enough for storing data before they are updated. So the detected frequent-updated data will be updated into the hot region (④). However, frequent-updated data may become infrequent with updates. Thus, when refresh operation is triggered, the valid data will be rewritten into the warm region for longer retention time (⑤).

Notably, there are data updated after a long retention time, especially in cold region. For read-intensive data, large read disturbance will be occurred on these data. For these data, Liu *et al.* [48] proposed to execute refresh operations if pages are read out too many times. The work is orthogonal to the proposed work.

Garbage Collection: As the three regions offer space for different kinds of data, thus GC of each region is conducted, respectively. For cold region, if there are no available space for new data, greedy GC [49] will be triggered to recycle more invalidated space. As the frequent-updated data will also be stored in the cold region first, after they are updated, the block with the most invalidated space will be selected for recycling. The warm region conducts similar GC with cold region. For hot region, if there are not enough space for new frequent-updated data, in-order GC is triggered to recycle the invalidated space. This is because that frequent-updated data will be updated in a small region, and all the update interval is almost less than 1 day, thus first-write-first-erase policy is suitable. During GC, if there are valid data, the data will be rewritten into the warm region as infrequent-updated data. With the separated GC design on each region, the block indexing cost on hot region can be moved.

Wear Leveling: In the work, dynamic wear leveling is adopted taking advantage of the request queue with minimized indexing cost. In the data and block matching approach, no matter in the cold, warm or hot regions, the blocks with the longest retention time will be allocated first. There are two reasons for applying dynamic wear leveling. First, PV can be exploited for lifetime improvement. As the retention time of flash blocks have been detected and maintained by the multiple-rank refresh queue, the related endurance of flash blocks is also exploited. That is, block with long supported retention time has large endurance. Thus, with dynamic wear leveling, using the longest-endurance flash block in priority can lengthen the lifetime of flash memory [26]. Second, if static wear leveling is applied, there will be many migration operations triggered to make the wearing of flash blocks evened, which is redundant and burdensome. In conclusion, dynamic wear leveling is applied for flash memory.

4.3.3 Lifetime Concern

However, there is a problem that allocating frequent-updated data to blocks with short retention time will make these blocks worn-out quickly. Thus, the lifetime of blocks in hot region will decide the lifetime of the overall flash memory. Compared with the improved lifetime with fully exploited PV, the improved lifetime will be weakened.

In order to solve this issue, we propose to set the hot region with a large ratio. The reasons come from two

aspects. First, with the large ratio hot region, most blocks in the hot region will be worn evenly with dynamic wear leveling. Thus, the lifetime of the flash memory will only be weakened with a little degree. Second, as shown in Figure 5(b), almost 90% of requests are updated within one day. In this way, large ratio region is allocated for large ratio requests, so that the three regions will be worn in a similar pace. Thus, the PV can be remained for refresh reduction, so that the performance can be improved over the whole lifetime of flash memory.

In detail, the region ratios of cold, warm and hot is set with 10%, 10% and 80% in this paper. Thus, the lifetime of flash memory will be approximately 80% of that with full-exploited PV.

4.4 Implementation and Overheads Analysis

In this section, the implementation is first presented. With the implementation, the overhead is then analyzed.

4.4.1 Implementation

Figure 9 shows the implementation of the proposed work. Based on the technology design above, there are three components should be implemented in the flash controller: (1) Retention Time Detector (RTD) for retention time detection, (2) Refresh Queueing (RQ) for fixed-period refresh under PV, (3) Block Allocator (BA) for data and block matching. In addition to these components, the unidirectional represents the dataflow among different components. Besides, there is a time trigger for each refresh rank in RQ, which is represented by a red button in the figure.

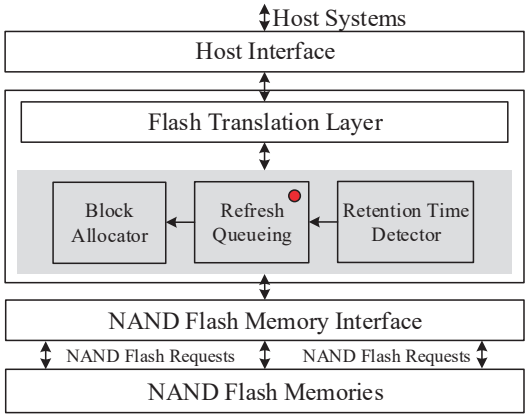


Fig. 9. Implementation structure of the proposed work in the flash memory controller.

Retention Time Detector: Retention time detector is designed to detect the supported retention time of each flash block. Here, multiple retention time are designated in advance. Each time, the longest retention time supported by the flash block is selected as the supported retention time. Initially, the longest retention time is regarded as the supported retention time. With the basic knowledge, retention time detection is to decide whether flash block can support the current retention time. The detection is conducted by comparing the maximum faulty bit count of block acquired from RQ with the threshold h_r and h_l described as shown in Figure 6. Thus, the current supported retention time of the block should be recorded in this component. Here, we designate one retention time as a refresh rank, and a refresh rank table is maintained. For the table, only one element is required, that is the refresh rank, as the refresh rank of each flash block should be maintained and the location can be as the index when checking in the table. After retention time detector, the refresh rank of each flash block is determined, which is required by the fixed-period refresh operation in RQ.

Refresh Queueing: Refresh queueing is designed to conduct fixed-period refresh for flash memory, where a multiple-rank queue is maintained. For each refresh rank, there is a queue maintaining blocks with the supported retention time. The construction of the multiple-rank queue requires the refresh rank of each flash block from RTD component. With the multiple-rank queue, the fixed-period refresh will be triggered regularly for each refresh rank, where the refresh interval should be set in advance. At the beginning of each refresh round, the refresh interval will be reset as there may be a little adjustment in each refresh queue. To get the refresh interval, the flash block number of each rank should been recorded. In this way, the corresponding refresh interval will be set for multiple-rank refresh queue respectively. Every each refresh interval, a refresh will be performed on a flash block. To conduct accurate refresh, the location information of the block is gotten from the queue, then read and rewrite operations are performed.

Block Allocator: Block allocator is designed to detect the update intervals of data and then allocate matching blocks for the data. As the supported retention time of flash blocks have been detected, with the refresh minimization goal, we only need to compare the supported retention time of flash blocks and the update interval of data to decide the match. That is if data are updated before refresh, blocks with smaller supported retention time are allocated for them. Otherwise, blocks with longer supported retention time are allocated. Three regions are divided to match the data and block. In each region, dynamic wear leveling is applied where blocks with the longest supported retention time will be allocated in priority.

4.4.2 Overhead Analysis

Area cost. The proposed scheme does not require hardware changes. It requires changes in FTL software/firmware to implement the three components. Besides, the faulty bit error number and comparison with threshold can take advantage of the existing flash systems to implement ECC and WL.

Space cost. There are space costs in software/firmware implementation. Assume that the configured 256GB flash memory has 4 channels, 8 chips per channel, 2 dies per chip, 2 planes per die, and 2048 blocks per plane in flash memory. Thus, the total number of flash block is 2^{18}, and 18 bits is needed to record the location. The memory and storage overheads of the proposed scheme are analyzed as follows. According to the detailed descriptions of components above, there are two big space required. The first is the refresh rank of each flash block. There are N refresh ranks, thus a *logN*-bit space is needed to record the refresh rank for each flash block. In RTD, maintaining the refresh rank for each flash block only requires $2^{18} * logN$ bits space. When N is set to 10, the space overhead is 128KB. While

in RQ, a multiple-rank queue requires space to record the block location. Thus, there are at most $18 * 2^{18}$ bits (576KB) space overhead. Besides, for multiple-rank queue, each rank maintains the flash block number, which is only several bits. In total, the space overhead is approximately 704KB, which is negligible for 256GB flash memory.

Response time effects. Refresh operations are redundant, having bad effects on response time. Expect refresh, other operations having few effects on response time which are negligible. First, the proposed scheme is aimed at reducing refresh for performance optimization. To further minimize the effects, the fixed-period refresh is applied in background. Second, the detailed response time are presented in the following section.

Power effects. First, to perform a refresh, the flash memory must be powered. As the scheme is proposed for enterprise storage applications, these systems are typically continuously powered on. Our proposed techniques use daily, weekly or monthly refresh and it is rare for a server to be powered off for such long periods. Second, the proposed scheme is to reduce refresh operations which will also reduce power consumption. The detailed power consumption is presented in the following section.

5 Experiments and Analysis

5.1 Methodology

To evaluate the supported retention time of flash blocks under PV, a supported retention time model under PV is first presented. Based on the model, the experiment setup is then presented.

5.1.1 Supported Retention Time Model under PV

A supported retention time model under PV is presented here for the evaluation of retention time detection approach. In traditional adaptive-rate refresh frequency method, the number of P/E cycles is the metric. However, with PV, the real endurance of flash blocks under some P/E cycles are different. Thus, in this model, the refresh frequency of flash blocks will be constructed by considering the real endurance of flash blocks.

Cai *et al.* [7] [10] verified that there is a relationship among raw bit error rate (RBER), wearings, and retention time, where RBER is proportional to the number of wearings and retention time and the measurement is conducted under room temperature (20°C). The wearings are determined by the P/E cycles to the blocks. Park *et al.* [21] then presented the relationship function RBER(*c*, *d*) as follows:

$$\begin{aligned} RBER(c,d) &= d_r(c) \cdot d \\ d_r(c) &= 10^{-13} \cdot c^{1.71} \end{aligned} \tag{1}$$

where $d_r(c)$ is the error deterioration rate per day, *c* is the number of wearings, and *d* is the retention time whose unit is day. Based on Equation 1, the supported retention time of a block under a specific number of wearings can be modeled as follows.

$$d = \frac{RBER_{th}}{d_r(c)} \tag{2}$$

where $RBER_{th}$ represents the error correction capability of the deployed ECC. Thus, the supported retention time is determined when the wearings of the block are determined.

Considering PV, the wearings of blocks induced by each P/E cycling are different from block to block. For example, for strong blocks, each P/E cycling induces small wearing, and vice-versa. Assume that flash blocks are worn linearly, the wearings of blocks can be modeled with the following formula:

$$c = P/E \cdot WD \tag{3}$$

where *P/E* is the number of P/E cycles, and *WD* is the wearing degree of each P/E cycling, which is a constant.

Besides, the temperature effect on supported retention time is usually described with the help of the Arrhenius law:

$$\frac{d(T_{ues})}{d(T_{reference})} = exp[\frac{E_{aa}}{K}(\frac{1}{T_{use}} - \frac{1}{T_{reference}})] \tag{4}$$

The higher the temperature (T_{use}) is, the lower the retention time (d_{use}) will be. In the Equation, T_{use} and $T_{reference}$ are absolute temperatures, *K* is the Boltzmann constant and E_{aa} is a process-dependent constant [38] [8].

Combining Equations (1) (2) (3) and (4), the supported retention time of flash blocks under one P/E cycles can be gotten. And the refresh frequency which is reciprocal to the supported retention time is also derived.

The supported retention time model is simple but effective. The goal of the model is to validate the efficiency of the proposed scheme where the relationship between PV and the supported retention time is concluded. Besides, the model can be extended with considering more factors, such as read disturbance [48].

5.1.2 Experiment Setup

In this section, a trace driven simulator, SSDsim [27], is used to verify the efficiencies of the proposed work. Table 2 lists the parameters of our simulated NAND flash-based SSD. The latencies (the first four rows of the table) are from the configurations in [23], while the sizes (rows 5-7) represent a modern commercial NAND flash specification [25].

TABLE 2
The parameters of the simulated flash-based SSD [23] [25].

Parameter	Value
Data bus latency	50 μs
Page read from register latency	75 μs
Page write to register latency	1.3 ms
Block erase latency	3.8 ms
Page/Block size	8KB/1MB
Die/Package size	8GB/64GB
Total storage capacity	256GB

Here, PV is assumed as a log normal distribution model as analyzed in [12], where the available P/E cycles of flash blocks are ranging at [15000, 24000], all higher than the one (10000) provided by industry. And with the supported retention time model under PV, the coefficient *WD* of each flash block in Equation 3 is set with the fraction between 10000 and the available P/E cycles. Figure 10 shows the relationship between the supported retention time and the P/E cycles of three kinds of flash blocks. As shown in the figure, *WD = 1* represents the original case without considering PV. While *WD = 1/1.5* represents the relationship of blocks whose available P/E cycles are 15000, and so on for *WD = 1/2.4*. Here, 11 refresh ranks shown in *x*-axis are configured as the available supported retention time in retention time

detection method. And block with *WD = 1/2.4* can support longer retention time than block with *WD = 1/1.5* at different P/E cycles.

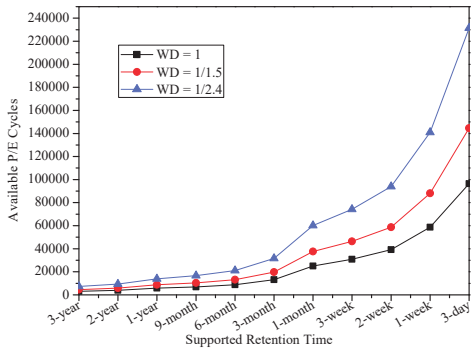


Fig. 10. The relationship of the supported retention time and P/E cycles of three kinds of flash blocks.

The proposed work is evaluated with 7 workload traces on the simulator. These traces have been widely used in many prior works [15] [21] [25] and all of them are random workloads, mixed with hot and cold data. The workload traces are all collected from MSR-Cambridge [28], and the tracing time is within 7 days. To evaluate the efficiency of the proposed work, four lifetime stages are measured, including 3-year, 1-year, 3-month and 3-week retention time for traditional reliable flash. Blocks under PV can support longer retention time due to *WD* compared with the traditional case. And we fill 50% of the usable space of the flash drive with data considering the normal use case. Similar to the approach employed in prior work [25], the overall lifetime is derived using the average write frequency of one run, which consists of writes generated by the trace and by garbage collection, as well as by refresh operations.

Here, we evaluate and compare among four schemes:

- ARFCR (adaptive rate based flash correct-and-refresh) is the baseline, which proposed to improve the lifetime by refreshing. In this scheme, it adopts different refresh frequencies for various lifetime stages without considering write hotness and PV [10].
- ARFCR+WARM (write-hotness aware retention management) is the recent work, which proposed to minimize refresh operations by identifying the write hotness. In this way, the refresh operation for hot data (frequent-updated data) can be skipped. In the scheme, two parameters including cooldown window size and hot pool are set based on the optimal one shown in [25].
- ARFCR+RTD (retention time detection) is the proposed retention time detection method, which only detects the different refresh frequencies under PV.
- ARFCR+Matching is the proposed data and block matching method, which allocates data to matched blocks with different refresh frequencies. The default ratios of cold, warm and hot regions are 10%, 10% and 80%, which can be represented by 1/1/8.

5.2 Experiment Results

In this subsection, the important results are presented and analyzed:

1) The decreased refresh operations with the proposed work are shown first.
2) With the minimized refresh count, the improved performances are then presented;
3) The effects on the lifetime of flash memory are also discussed;
4) Finally, the details and the evaluation of the scheme are presented.

5.2.1 Reduced Refresh Count

Figure 11 shows the refresh count among the four schemes over four lifetime stages. As shown in the figure, with the wearing of flash memory, the refresh count is increased with reduced refresh period for each trace. However, with the four schemes, the original increased refresh counts are reduced in different degrees. Compared to the baseline ARFCR, ARFCR+WARM reduces less refresh count for each stage, the reduction ratio is 5% for all stages on average, fitting the results in [25]. For ARFCR+RTD, the reduction ratio is about 63.3%, 72.4% and 35.8%, respectively for 1-year, 3-month, and 3-week stages. Furthermore, for ARFCR+Matching, the reduction ratio is increased to 66.2%, 81.2% and 58.8%, respectively for 1-year, 3-month, and 3-week stages.

The reduction ratio of ARFCR+WARM is low, and the reasons are from two aspects. First, according to Figure 5(b), almost 90% of requests are updated within 1 day, thus the situation of storing hot and cold data in a block is rare. Second, the hot and cold separation method is blind to the retention of flash blocks. While the proposed ARFCR+RTD exploits the real endurance of flash blocks under PV, which is an inherent characteristic. For the proposed matching method, the benefits of the prior three stages are also small. But the benefits are still little better than that of ARFCR+WARM, as the method is guided by the supported retention time of flash blocks. While at the 3-week stage, the reduction is large, this is because lots of refresh have been skipped. In detail, the cold data are all put into strong blocks with long supported retention time and the short supported retention time of weak blocks can still satisfy the hot data.

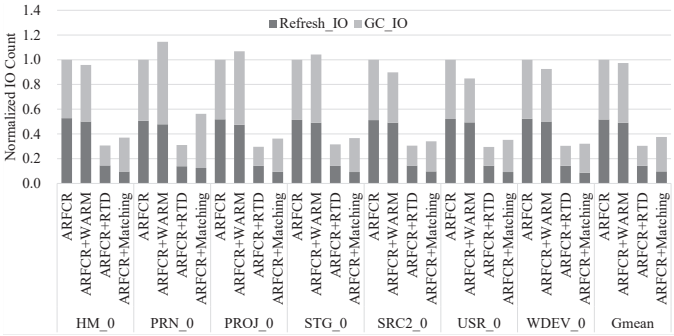


Fig. 12. Redundant operations in flash memory for the four schemes when the stage is at 3 months.

5.2.2 Redundant Operations

Before introducing the performance and lifetime, we first present the redundant operations in flash memory for the four schemes over the 3-month stage. Though the refresh operations are reduced, there are also other internal operations (GC) in flash memory, which will have influence

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2858771, IEEE Transactions on Computers

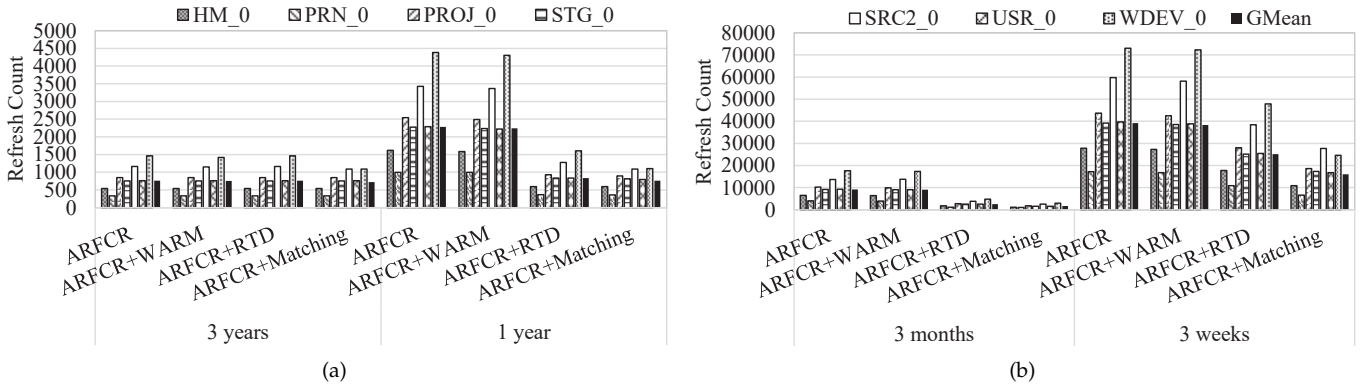
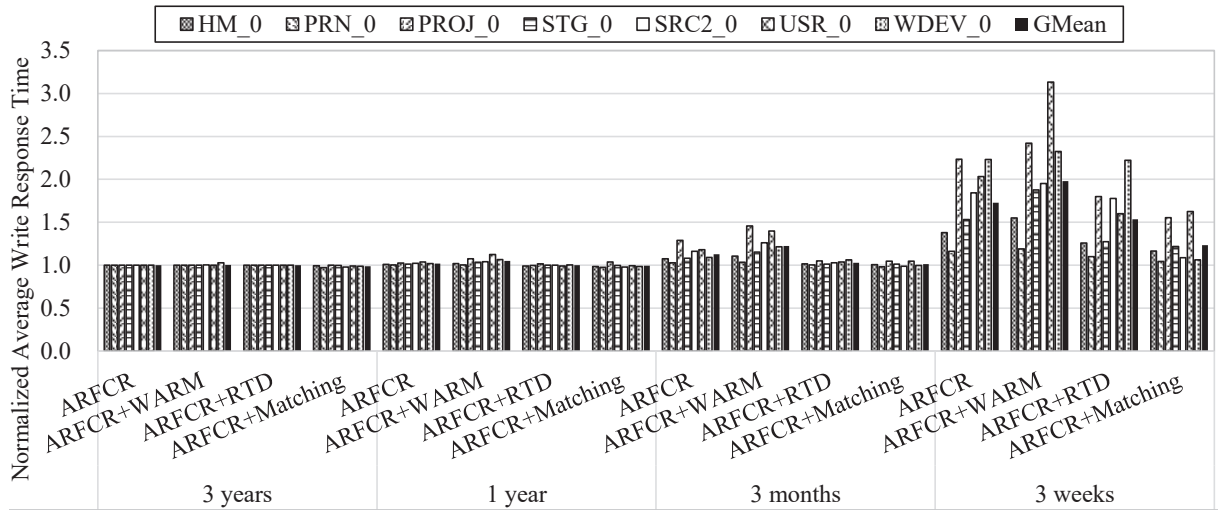
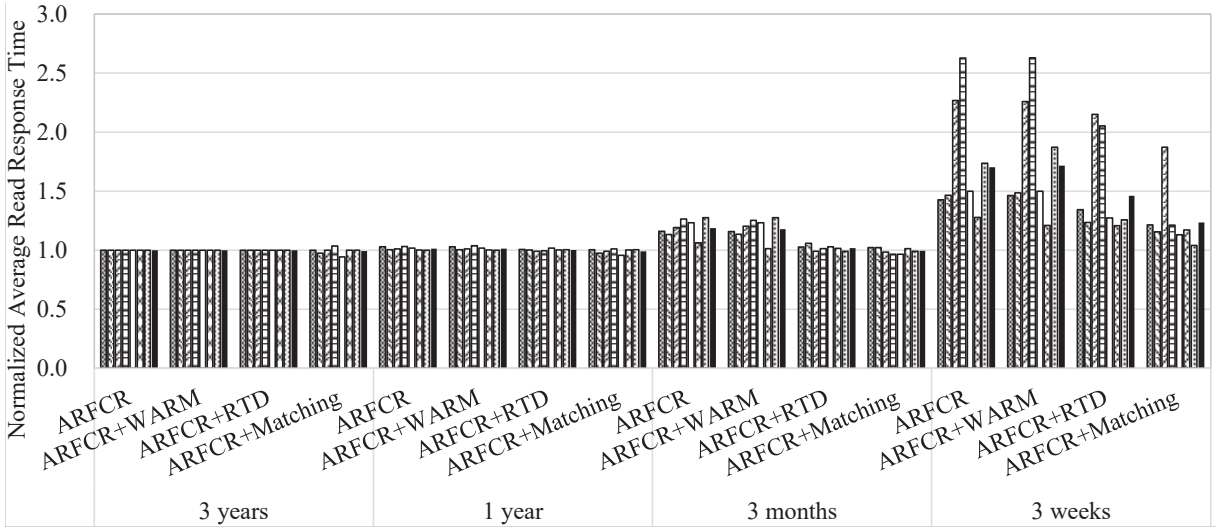


Fig. 11. The number of refresh comparison among ARFCR, ARFCR+WARM, ARFCR+RTD and ARFCR+Matching.



(a) Average write response time.



(b) Average read response time.

Fig. 13. Average write and read response time comparison among ARFCR, ARFCR+WARM, ARFCR+RTD and ARFCR+Matching schemes.

on the performance and lifetime of flash memory. Figure 12 presents the normalized IO count of the four schemes, including Refresh IOs and GC IOs, where the baseline is the total count of ARFCR platform. From the figure, we can find that compared to ARFCR, ARFCR+WARM introduces more GC operations, this is due to the limited hot pool size. As hot requests are updated frequently, GC will also be triggered frequently for more space. Besides, ARFCR+Matching introduces more GC operations compared to ARFCR+RTD, where the reason is similar also due to the limited region ratios. Notably, for different stages, the GC operations will be similar as the simulated workloads are same.

5.2.3 Effects on Performance

Since refresh operations are redundant operations, reducing refresh operations is good to the performance. As shown in Figure 13, the average write and read response time are both measured for the four schemes under four stages. All average response time have been normalized to the average response time with 3-year refresh frequency in ARFCR platform. Besides, the table in the figure presents the baseline of the average write/read response time.

From Figure 13, we have the following observations: First, ARFCR+WARM performs little worse write performance than ARFCR, where the write latency overhead in the first 3 stages is within 5%, but increased to 15% at the 3-week stage. For read performance, there are only within 1% performance overhead. The performance effects are similar to that in [25]. The reason is that though the refresh operations are reduced, more GC operations are introduced due to small hot pool. While the performance of the ARFCR+RTD and ARFCR+Matching schemes perform better. For write performance, ARFCR+RTD reduces the latency by 1.58%, 6.74% and 6.22%, respectively for 1-year, 3-month and 3-week stages. ARFCR+Matching reduces the latency by 0.9%, 10% and 33.18%, respectively. Similar to the write performance, the read performance is also improved with the proposed methods. Compared to ARFCR, the average read performance of ARFCR+RTD is improved with 0.1%, 12.28% and 11.35%, respectively for 1-year, 3-month and 3-week stages. ARFCR+Matching shows better read performance, where the improved ratio is 0.5%, 16.8% and 30.2%, respectively. This is because that lots of unnecessary refresh operations are reduced for ARFCR+RTD platform. Though there are also GC operations triggered in the three regions for ARFCR+Matching platform, more unnecessary refresh operations are reduced. Besides, with background GC, the performance effects become lower. Thus, the performance are improved with the proposed scheme.

5.2.4 Effects on Lifetime

Figure 14 shows the normalized lifetime of four schemes. Compared with the baseline ARFCR, other three schemes all achieve the lifetime improvement. In detail, ARFCR+WARM improves the lifetime by 15.5%. Though the reduced refresh count for every stage is low, the total lifetime improvement is the accumulated result. Through exploiting PV, ARFCR+RTD improves the lifetime by 213.3%. This is because the total endurance of all blocks are fully exploited without unnecessary refresh operations. With the 1/1/8 region ratio, the lifetime of ARFCR+Matching is smaller than that of ARFCR+RTD by 23.3%. The result coincides with the hot region ratio 80%, where the total endurance of blocks in warm and cold regions are not fully exploited. As lifetime concern analysis in Section 4.3.2, the lifetime of flash memory will only be weakened with a little degree. While compared with the baseline, the improved lifetime is still achieved by 163.7%.

5.2.5 Evolution of PV

Here, the default ratios of the cold, warm and hot regions in matching method are 10%, 10% and 80%. To evaluate the evolution of PV, we measured the request ratios under the four stages, as shown in Figure 15. As shown in the figure, most requests are hot requests, and allocated to hot region. While the cold ratio is floated around 10%, and warm ratio is lower than 10%. With the reducing of refresh period, the cold ratio is increased to larger than 10% while hot is decreased and warm is remained. As the matching method designed, the warm region can be used to sustain cold region or hot region. Combining with the default region ratios, the PV distribution will be remained as the wearing on each region is synchronous. Further, according to the average request ratios at each stage, we measured the available P/E cycles distribution of flash blocks after the four stages as shown in Figure 16. As shown in the figure, the distribution after each stage is similar to the original one. In this way, the benefits on refresh minimization can be achieved.

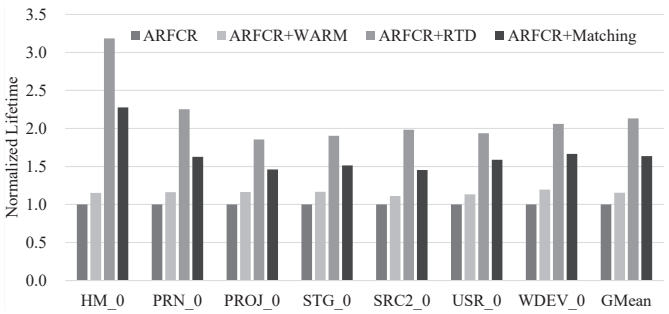


Fig. 14. Lifetime comparison among ARFCR, ARFCR+WARM, ARFCR+RTD and ARFCR+Matching under different workload traces.

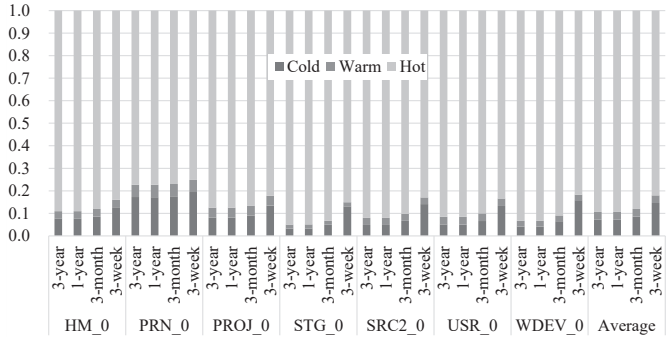


Fig. 15. Request ratios among the four stages under different workload traces.

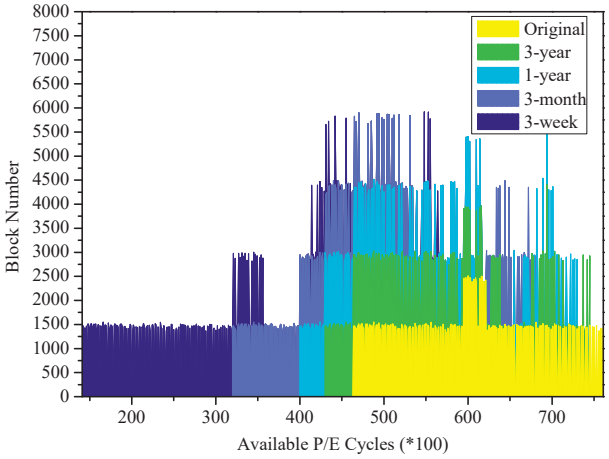


Fig. 16. The available P/E cycles distribution of flash blocks after the four stages.

5.2.6 Power Consumption

For ARFCR platform, the power consumption for entire SSD refresh within three weeks is 0.37% [10]. We assume that the power consumption is proportional to the operation number, including refresh and GC operations. The more operation number is, the more power will be consumed. Combing Figure 11 and 12, the power consumption is shown in Table 3. As shown in the figure, the power consumption is the average power consumption for the seven traces at the four platforms. Besides, the power consumption of the proposed scheme is reduced with refresh minimization.

TABLE 3
The power consumption under the four platforms.

Platforms	3 years	1 year	3 months	3 weeks
ARFCR	0.072%	0.084%	0.138%	0.37%
ARFCR+WARM	0.072%	0.083%	0.134%	0.362%
ARFCR+RTD	0.028%	0.029%	0.042%	0.217%
ARFCR+Matching	0.044%	0.044%	0.052%	0.162%

5.2.7 Benefits of the Matching Approach

In Di *et al.* [1], size threshold is used as the metric on the hot and cold identification. The identification approach also can cooperate with ARFCR+RTD platform for further reducing refresh operations. Figure 17 presents the comparison between the ARFCR+Matching platform and Di *et al.* under 3-month refresh frequency, where 16, 32, 48 and 64 are respectively as the size thresholds for hot and cold identification. From the figure, there are several findings. First, Di *et al.* can also achieve the goal of further reducing the refresh operations. Besides, with different size thresholds, the benefits on the refresh count reduction and the optimal size threshold for each trace are both different. Second, the proposed ARFCR+Matching platform performs better than Di *et al.* significantly. This is because that the proposed matching approach fully exploits the relationship between update interval of data and the supported retention time of flash blocks. In average, Di *et al.* can reduce 14.8% refresh count compared with ARFCR+RTD platform. While the ARFCR+Matching platform can further reduce 20.6% refresh counts compared with Di *et al.*.

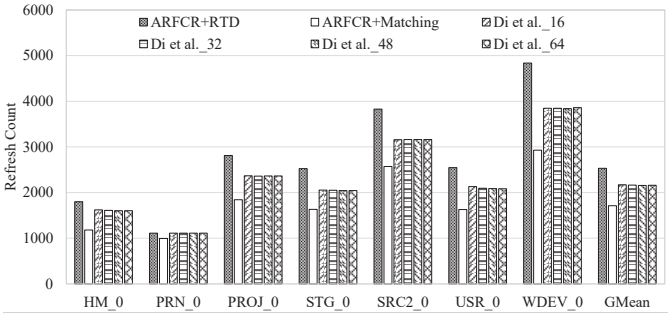


Fig. 17. The comparison between the ARFCR+Matching platform and Di *et al.* under 3-month refresh frequency.

5.2.8 Temperature Effects

In this subsection, the refresh counts under two different temperatures are measured. The default temperature is 20°C, while temperature becomes smaller to 15°C with minimized refresh operations. Besides, the supported retention time of flash block is assumed with 3 months with default temperature. According to Equation 4 and linear extrapolation with values in [8], the supported retention time of flash blocks are increased with about 5 times under 15°C temperature. Figure 18 shows the refresh count under different temperatures. With smaller temperature, refresh counts are reduced significantly as the refresh frequency is decreased with increasing retention time. Besides, the ARFCR+RTD benefit under 15°C is 66.6% slightly smaller than that under 20°C (72.4%). This is because that the retention time differences among flash blocks are slightly decreased with 5-time changing. In addition, the ARFCR+Matching benefit is similar to ARFCR+RTD benefit under 15°C as all flash block can support enough long retention time.

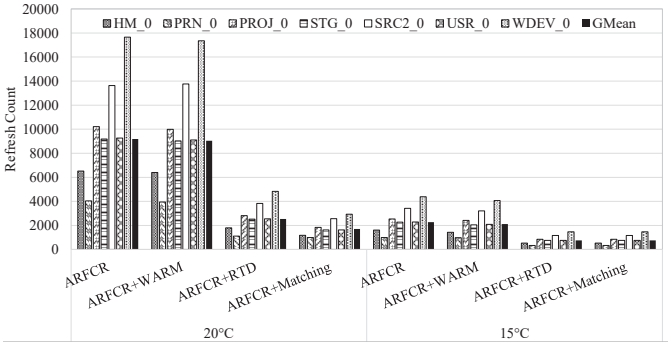


Fig. 18. The refresh count minimization under different temperatures.

6 Conclusion

With technology scaling and density improvement, the reliability of flash memory is decreasing. To solve the challenge, refresh is proposed to improve the reliability. However, refresh is a redundant operation, and will hurt the performance and lifetime of flash memory. Different from recent work, this work is proposed to minimize the refresh operations considering PV. The work is motivated by two findings. The first one is that the real endurance among blocks are different and higher than the thought endurance under some P/E cycles. The second one is that the update intervals of data are different, some long, some short. Combining the two findings, the basic idea is to exploit the supported retention time of flash blocks, and allocate data to blocks with higher supported retention time, so that the refresh operations can be minimized. In this work, two methods, refresh frequency determination and data and block matching method, are designed. With the proposed work, implementation for the overall work is presented, and the overhead analysis shows that the space overhead is negligible. Series of simulations are evaluated, and from the experiment results, the PV aware scheme on retention induced refresh minimization significantly improves performance and lifetime of flash memory.

7 Acknowledgement

This work is supported by the NSFC (61772092 and 61572411).

References

[1] Y. Di, L. Shi, K. Wu, and C. J. Xue, "Exploiting process variation for retention induced refresh minimization on flash memory," in *DATE*, March 2016, pp. 391–396.

[2] M. Helm, J.-K. Park, A. Ghalam, J. GUO, C. wan Ha, C. Hu, H. Kim, K. Kavalipurapu, E. Lee, A. Mohammadzadeh, D. Nguyen, V. Patel, T. Pekny, B. Saiki, D. Song, J. Tsai, V. Viajedor, L. Vu, T. Wong, J. H. Yun, R. Ghodsi, A. d'Alessandro, D. Di Cicco, and V. Moschiano, "19.1 a 128gb mlc nand-flash device using 16nm planar cell," in *ISSCC*, Feb 2014, pp. 326–327.

[3] S. Choi, D. Kim, S. Choi, B. Kim, S. Jung, K. Chun, N. Kim, W. Lee, T. Shin, H. Jin, H. Cho, S. Ahn, Y. Hong, I. Yang, B. Kim, P. Yoo, Y. Jung, J. Lee, J. Shin, T. Kim, K. Park, and J. Kim, "19.2 a 93.4mm2 64gb mlc nand-flash memory with 16nm cmos technology," in *ISSCC*, Feb 2014, pp. 328–329.

[4] K.-C. Ho, P.-C. Fang, H.-P. Li, C.-Y. Wang, and H.-C. Chang, "A 45nm 6b/cell charge-trapping flash memory using ldpc-based ecc and drift-immune soft-sensing engine," in *ISSCC*, Feb 2013, pp. 222–223.

[5] Y. H. Chang, P. C. Huang, P. H. Hsu, L. J. Lee, T. W. Kuo, and D. H. C. Du, "Reliability enhancement of flash-memory storage systems: An efficient version-based design," *IEEE TC*, vol. 62, no. 12, pp. 2503–2515, Dec 2013.

[6] M. C. Yang, Y. H. Chang, and T. W. Kuo, "Virtual flash chips: Reinforcing the hardware abstraction layer to improve data recoverability of flash devices," *IEEE TC*, vol. 65, no. 9, pp. 2872–2883, Sept 2016.

[7] Y. Cai, E. Haratsch, O. Mutlu, and K. Mai, "Error patterns in mlc nand flash memory: Measurement, characterization, and analysis," in *DATE*, March 2012, pp. 521–526.

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

[9] H. So and S. Wong, "Multi-bit-per-cell flash eeprom memory with refresh," Nov. 21 2000, uS Patent 6,151,246.

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

[11] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: Improving nand flash lifetime by balancing page endurance," in *USENIX FAST*, Santa Clara, CA, 2014, pp. 47–59.

[12] Y. Pan, G. Dong, and T. Zhang, "Error rate-based wear-leveling for nand flash memory at highly scaled technology nodes," *TVLSI*, vol. 21, no. 7, pp. 1350–1354, July 2013.

[13] Y.-J. Woo and J.-S. Kim, "Diversifying wear index for mlc nand flash memory to extend the lifetime of ssds," in *EMSOFT*, Sept 2013, pp. 1–10.

[14] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," *ACM SIGMETRICS*, vol. 43, no. 1, pp. 177–190, June 2015.

[15] L. Shi, Y. Di, M. Zhao, C. Xue, K. Wu, and E.-. Sha, "Exploiting process variation for write performance improvement on nand flash memory storage systems," *TVLSI*, vol. 24, no. 1, pp. 334–337, January 2016.

[16] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. Long, A. Hospodor, and S. Ng, "Disk scrubbing in large archival storage systems," in *IEEE MASCOTS*, 2004, pp. 409–418.

[17] I. Bhati, Z. Chishti, S. Lu, and B. Jacob, "Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions," in *ISCA*, 2015, pp. 235–246.

[18] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *HPCA*, Feb 2012, pp. 1–12.

[19] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," ser. IEEE ISCA, Washington, DC, USA, 2012, pp. 1–12.

[20] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "Chargecache: Reducing dram latency by exploiting row access locality," in *IEEE HPCA*, March 2016, pp. 581–593.

[21] H. Park, J. Kim, J. Choi, D. Lee, and S. H. Noh, "Incremental redundancy to reduce data retention errors in flash-based ssds," in *MSST*, May 2015, pp. 1–13.

[22] L. Shi, K. Wu, M. Zhao, D. Liu, J. Xue, and E. Sha, "Retention trimming for lifetime improvement of flash memory storage systems," *TCAD*, vol. PP, no. 99, pp. 1–1, 2015.

[23] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing nand flash-based ssds via retention relaxation," *Target*, vol. 11, no. 10, p. 00, 2012.

[24] V. Mohan, S. Sankar, S. Gurumurthi, and W. Redmond, "refresh ssds: Enabling high endurance, low cost flash in datacenters," *Univ. of Virginia, Tech. Rep. CS-2012-05*, 2012.

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

[26] M.-C. Yang, Y.-H. Chang, C.-W. Tsao, and P.-C. Huang, "New era: new efficient reliability-aware wear leveling for endurance enhancement of flash storage devices," in *ACM DAC*, 2013, p. 163.

[27] 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 *ACM ICS*, New York, NY, USA, 2011, pp. 96–107.

[28] "Snia: Iotta reponsitory, http://iotta.snia.org/tracetypes/3," http://iotta.snia.org/tracetypes/3.

[29] C. Kim, C. Park, S. Yoo, and S. Lee, "Extending lifetime of flash memory using strong error correction coding," *IEEE TCE*, vol. 61, no. 2, pp. 206–214, May 2015.

[30] S. Lee and J. Kim, "Effective lifetime-aware dynamic throttling for nand flash-based ssds," *IEEE TC*, vol. 65, no. 4, pp. 1075–1089, April 2016.

[31] R. Micheloni, L. Crippa, and A. Marelli, "Inside nand flash memories," in *Springer Dordrecht Heidelberg London New York*, 2010.

[32] 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 TECS*, vol. 3, no. 4, pp. 837–863, 2004.

[33] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design," in *ACM DAC*, 2007, pp. 212–217.

[34] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile ssd: Trading flash memory nonvolatility to improve storage system performance for enterprise applications," in *HPCA*, Feb 2012, pp. 1–10.

[35] J. Jeong, Y. Song, S. S. Hahn, S. Lee, and J. Kim, "Dynamic erase voltage and time scaling for extending lifetime of nand flash-based ssds," *IEEE TC*, vol. 66, no. 4, pp. 616–630, April 2017.

[36] Q. Wu and T. Zhang, "Ofwar: Reducing ssd response time using on-demand fast-write-and-rewrite," *IEEE TC*, vol. 63, no. 10, pp. 2500–2512, Oct 2014.

[37] H. Wang, N. Wong, T. Y. Chen, and R. D. Wesel, "Using dynamic allocation of write voltage to extend flash memory lifetime," *IEEE TC*, vol. 64, no. 11, pp. 4474–4486, Nov 2016.

[38] M. Seif, E. Farjallah, F. Badets, E. Chabchoub, C. Layer, J. M. Armani, F. Joffre, C. Anghel, L. Dilillo, and V. Gherman, "Refresh frequency reduction of data stored in SSDs based on A-timer and timestamps," in *IEEE ETS*, May 2017, pp. 1–6.

[39] Y. Du, Q. Li, L. Shi, D. Zou, H. Jin, and C. J. Xue, "Reducing LDPC soft sensing latency by lightweight data refresh for flash read performance improvement," in *ACM/EDAC/IEEE DAC*, June 2017, pp. 1–6.

[40] I. Bhati, M. T. Chang, Z. Chishti, S. L. Lu, and B. Jacob, "DRAM Refresh Mechanisms, Penalties, and Trade-Offs," *IEEE TC*, vol. 65, no. 1, pp. 108–121, Jan 2016.

[41] K. Zhou, W. Liu, K. Tang, P. Huang, and X. He, "Alleviating Memory Refresh Overhead via Data Compression for High performance and Energy Efficiency," *IEEE TPDS*, pp. 1–1, 2017.

[42] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," ser. IEEE DSN, Washington, DC, USA, 2015, pp. 427–437.

[43] S. Salehi, N. Khoshavi, and R. F. Demara, "Mitigating Process Variability for Non-Volatile Cache Resilience and Yield," *IEEE TETC*, pp. 1–1, 2018.

[44] D. Park and D. Du, "Hot data identification for flash memory using multiple bloom filters," *MSST*, pp. 1–11, 2011.

[45] J. Lee and J.-S. Kim, "An empirical study of hot/cold data separation policies in solid state drives (ssds)," ser. ACM SYSTOR, New York, NY, USA, 2013, pp. 12:1–12:6.

[46] L.-P. Chang, "Hybrid solid-state disks: combining heterogeneous nand flash in large ssds," in *IEEE ASPDAC*, 2008, pp. 428–433.

[47] A. Arreghini, G. V. den bosch, G. S. Kar, and J. V. Houdt, "Ultimate Scaling Projection of Cylindrical 3D SONOS Devices," in *IEEE IMW*, May 2012, pp. 1–4.

[48] C.-Y. Liu, Y.-M. Chang, and Y.-H. Chang, "Read Leveling for Flash Storage Systems," in *ACM SYSTOR*, New York, NY, USA, 2015, pp. 5:1–5:10.

[49] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," ser. USENIX ATC, Berkeley, CA, USA, 2008, pp. 57–70.

Yejia Di received BS degree in Computer Science and Technology from Chongqing University in China in 2014. She is now a Ph.D candidate in the College of Computer Science, Chongqing University. Her research interests include embedded and real-time systems, non-volatile memory and architecture optimizations.

Liang Shi received the B.S. degrees in Computer Science from Xi'an University of Post & Telecommunication, Xi'an, Shanxi, China, in July, 2008, Ph.D. degree from University of Science and Technology of China, Hefei, China, and Joint Ph.D. degree from City University of Hong Kong, Hong Kong, in June, 2013. He is now an associate professor in the College of Computer Science at the Chongqing University. His research interests include flash memory, embedded systems, and emerging non-volatile memory technology.

Congming Gao received BS degree in Computer Science and Technology from Chongqing University in China in 2014. He is now a Ph.D candidate in the College of Computer Science, Chongqing University. His research interests include embedded and real-time systems, non-volatile memory and architecture optimizations.

Qiao Li received the B.S. and M.S. degrees in Computer Science and Technology from Chongqing University in China in 2014 and 2017 respectively. She is now a PhD candidate in Department of Computer Science, City University of Hong Kong. Her research interests include NAND flash memory, embedded systems and computer architecture.

Chun Jason Xue received BS degree in Computer Science and Engineering from University of Texas at Arlington in May 1997, and MS and PhD degree in computer Science from University of Texas at Dallas, in Dec 2002 and May 2007, respectively. He is now an Associate Professor in the Department of Computer Science at the City University of Hong Kong. His research interests include memory and parallelism optimization for embedded systems, software/hardware co-design, real time systems and computer security.

Kaijie Wu received the BE degree from Xidian University, Xi'an, China, in 1996, the MS degree from the University of Science and Technology of China, Hefei, China, in 1999, and the Ph.D. degree in electrical engineering from Polytechnic University (Now Polytechnic Institute of New York University), Brooklyn , New York, in 2004. He then joined University of Illinois, Chicago, USA as an Assistant Professor. Since 2013, he becomes a professor at the College of Computer Science, Chongqing University, China. His research is on the big area of trustworthy computing with special interest on dependable computing and hardware security. He is the recipient of the 2004 EDAA Outstanding Dissertation Award for new directions in circuit and system test., and the Most Significant Paper award from the International Test Conference, 2014.