Heating Dispersal for Self-Healing NAND Flash Memory

Renhai Chen, Yi Wang, Duo Liu, Zili Shao and Song Jiang

Abstract—Substantially reduced lifetimes are becoming a critical issue in NAND flash memory with the advent of multi-level cell and triple-level cell flash memory. Researchers at Macronix have recently discovered that heating can cause worn-out NAND flash cells to become reusable and greatly extend the lifetime of flash memory cells. However, the heating process consumes a substantial amount of power, and some fundamental changes are required for existing NAND flash management techniques. In particular, all existing wear-leveling techniques are based on the principle of evenly distributing writes and erases. For self-healing NAND flash, this may cause NAND flash cells to be worn out in a short period of time. Moreover, frequently healing these cells may drain the energy quickly in battery-driven mobile devices, which is defined as the concentrated heating problem.
In this paper, we propose a novel wear-leveling scheme called DHeating (Dispersed Heating) to address the problem. In DHeating, rather than evenly distributing writes and erases over a time period, write and erase operations are scheduled on a small number of flash memory cells at a time, so that these cells can be worn out and healed much earlier than other cells. In this way, we can avoid quick energy depletion caused by concentrated heating. In addition, the heating process takes several seconds and has become the new performance bottleneck. In order to address this issue, we propose a lazy heating repair scheme. The lazy heating repair scheme can ease the long time heating effect by delaying the heating operation and using the system idle time to repair. Furthermore, the flash memory's reliability becomes worse with the flash memory cells reaching the excepted worn-out time. We propose an early heating strategy to solve the reliability problem. With the extended lifetime provided by self-healing, we can trade some lifetimes for reliability. The idea is to start the healing process earlier than the expected worn-out time. We evaluate our scheme based on an embedded platform. The experimental results show that the proposed scheme can effectively prolong the consecutive heating time interval, alleviate the long time heating effect, and enhance the reliability for self-healing flash memory.

Index Terms—Flash memory, self-healing, wear leveling, power consumption, dispersed heating.

1 Introduction

NAND flash memory has many advantages such as non-volatility, low power consumption, and good shock resistance. It has been widely used as storage devices in embedded systems. However, NAND flash memory has some constraints, particularly, limited lifetime. For example, multi-level cell (MLC) flash memory, which is the mainstream NAND flash product on the market, can withstand 10,000 program/erase (P/E) cycles; triple-level cell (TLC) flash memory, which is the emerging flash memory product, can withstand only 2,500 P/E cycles [1], [2], [3], [4], [5], [6]. In order to overcome this constraint, Researchers at Macronix recently invented self-healing flash memory, in which worn-out flash memory cells can be rejuvenated by thermal annealing [7].

- *Renhai Chen and Zili Shao are with Embedded Systems and CPS Laboratory, Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong.*
- *Yi Wang is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China.*
- *Duo Liu is with College of Computer Science, Chongqing University, Chongqing, China.*
- *Song Jiang is with Department of Electrical and Computer Engineering, Wayne State University, Detroit, USA.*
- *The corresponding author: Zili Shao (Email: cszlshao@comp.polyu.edu.hk; Tel: (852)27667287).*

However, heating a self-healing flash memory cell does consume a substantial amount of power [8]. If a large number of flash memory cells are heated in a short time period, the energy will be exhausted in a battery-driven embedded system such as smartphones. For traditional NAND flash memory, wear-leveling strategies are employed in an attempt to evenly distribute write and erase operations [9], [10], [11], [12], [13], [14]. If such strategies are used to manage self-healing flash memory, it will cause the concentrated heating problem, that is, when all flash memory cells wear out together, healing these cells in a concentrated manner might drain the energy. This paper focuses on solving this problem.

Several challenging issues should be considered when self-healing flash memory is used in embedded systems. First, the lifetime of self-healing flash memory is prolonged by the heating of worn-out flash memory cells. However, heating does not come out without a cost, as heating shortens the battery life and a heating operation consumes a large amount of power [8]. As a result, heating should not occur frequently, and only a small portion of flash memory cells should be healed at a time. Second, heated cells can be treated as newborn cells rather than as permanently retired cells. Since the reliability of newborn cells is better than that of nearly worn-out flash memory

cells, data should be moved from nearly worn-out cells to newborn cells. As discussed above, previous wear-leveling strategies are not suitable for use on this emerging self-healing NAND flash memory. Thus, it is very important to design a new management strategy for self-healing flash memory.

In this paper, we first propose a novel wear-leveling scheme, called DHeating, to manage self-heating NAND flash memory. DHeating performs wear leveling by storing frequently updated data, called hot data, in a small number of flash memory cells. By applying this technology to self-healing flash memory, heating operations are dispersed so that only a small portion of the cells will be healed each time, and the time interval for the consecutive heating operations is prolonged. In addition, the heating procedure takes several seconds, which becomes the new performance bottleneck in self-healing flash memory. In order to ease the long time heating effect, we propose a lazy heating repair scheme. The basic idea of the lazy heating repair scheme is to delay the heating operation and to repair during the system idle time period. Furthermore, the flash memory's reliability degrades dramatically with self-healing flash memory reaching its worn out point [15], [16], [17]. In order to enhance the reliability of the self-healing flash memory, we propose an early heating scheme in which we start to heal flash memory cells earlier than their expected endurance. To the best of our knowledge, this is the first work to address the concentrated heating problem through the use of a dispersed heating strategy, to ease the long time heating effect with a lazy heating repair technique, and to enhance the reliability of the self-healing flash memory by adopting an early heating scheme.

We have conducted experiments with various applications based on an embedded system with an ARM11 processor and an 8Gb NAND flash memory chip. The experimental results show that DHeating not only addresses the concentrated heating problem, but also improves the system response time in self-healing flash memory compared with the baseline scheme. The improvement of the system response time is mainly caused by reducing the swapping of hot and cold data that occurs frequently in previous wear-leveling strategies.

The main contributions of this work are:

- This work addresses the concentrated heating problem for self-healing NAND flash memory from the wear-leveling perspective.
- A lazy heating repair scheme is proposed to alleviate the long time heating effect.
- An early heating strategy is proposed to enhance the reliability of the self-healing flash memory.

The rest of this paper is organized as follows: Section 2 presents the background of this work and our motivation on conducing this work. Section 3 introduces our DHeating strategy with performance and overhead analysis. The experimental results are presented and discussed in Section 4. In Section 5, we conclude our work.

2 Background and Motivation

2.1 NAND Flash Memory and Worn Out

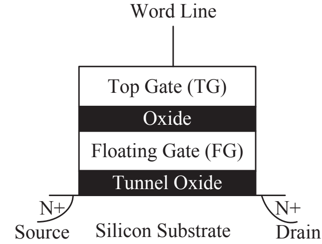


Fig. 1. The structure of a traditional NAND flash memory cell.

In a traditional NAND flash memory chip, a floating-gate transistor is used as the storage cell as shown in Fig. 1. A floating-gate transistor is made of one top gate (or control gate), one floating gate, two oxide layers, and the silicon substrate. The top gate, floating gate, and silicon substrate are surrounded by the oxide, which functions as insulation. The oxide layer between a floating gate and the silicon substrate is called *tunnel oxide*. When a high voltage is applied to a word line, which is connected to the top gate of a cell, the electrons from the silicon substrate will traverse the tunnel oxide and be trapped in a floating gate. The number of electrons trapped in a floating gate is used to denote the stored data [18], [19], [20].

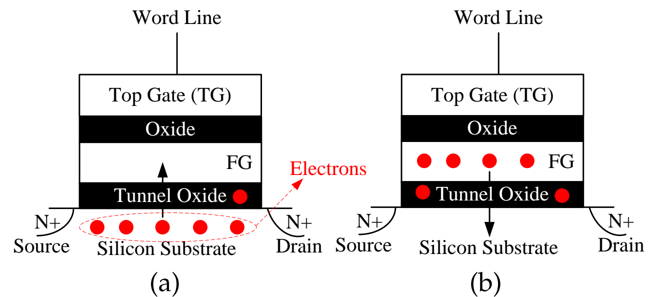


Fig. 2. Illustration on wearing of a flash memory cell caused by (a) the programming operation, and (b) the erase operation.

The lifetime of a flash memory cell decreases with constant program and erase operations, since repeated P/E cycles can damage the tunnel oxide layer as shown in Fig. 2. Program operations can cause electrons to traverse through the tunnel oxide layer to the floating gate as shown in Fig. 2 (a), while erase operations make electrons to traverse through the tunnel oxide layer to the silicon substrate as shown in Fig. 2 (b). The repeated traverse procedures may cause electrons to be trapped in the tunnel oxide layer.

Eventually, the tunnel oxide layer is damaged because of too many electrons being trapped in the tunnel oxide layer.

2.2 Self-Healing NAND Flash Memory and Heating Repair

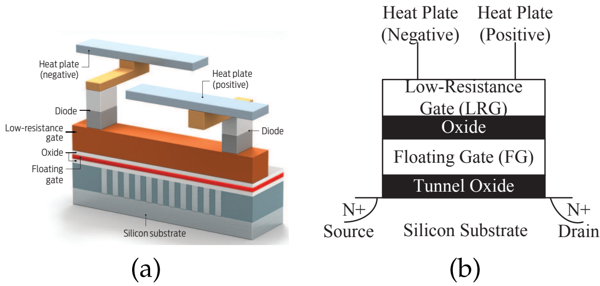


Fig. 3. The structure of a self-healing NAND flash memory storage cell. (a) 3D vision. (b) 2D vision.

Several state-of-the-art works [8], [21], [22] discover that high temperature can make electrons to be de-trapped from the oxide layer. Therefore, researchers at Macronix modify the traditional NAND flash memory structure and invent self-healing NAND flash memory. The structure of a self-healing flash memory cell is shown in Fig. 3(a) and Fig. 3(b). In self-healing NAND flash, a word line is modified to become a double-ended structure with one positive heat plate and one negative heat plate; a low-resistance gate is used as the top gate, which enables the current to pass through the gate [8]. The two heat plates are connected to a double-ended word line-one plate on each end.

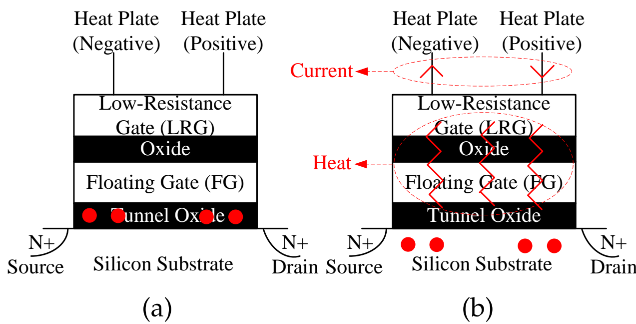


Fig. 4. A comparison of worn out flash memory and healed flash memory. (a) The worn out flash memory storage cell. (b) The healed flash memory cell.

If a flash memory cell reaches its lifetime as shown in Fig. 4 (a), a certain level of voltage will be applied to the corresponding heat plates and a high temperature (>800°C) will be generated immediately after the current passes through the gate as Fig. 4 (b) shown. Heating can repair the damaged tunnel oxide of the cell, thus extending the lifetime of the cell.

Since self-healing NAND flash memory extends its lifetime by heating damaged tunnel oxide layers rather than adding new tunnel oxide layers, a flash memory cell will permanently retire after several heating repairs. We assume that a self-healing NAND flash memory cell can be healed at most θ times, and after $i's$ $(1 \leq i \leq \theta)$ healing, it can be used L_i times. The lifetime of a self-healing flash cell can be represented as follows:

$$Lifetime = \sum_{i=0}^{\theta} L_i \quad (1)$$

In Equation 1, L_0 denotes the factory lifetime or the lifetime prior to healing, and $L_i(i\ \epsilon[1,\theta])$ denotes the lifetime in each stage.

Suppose that we combine this self-healing technology with TLC NAND flash memory. Typically the factory lifetime of a TLC NAND flash memory chip is 2,500 or L_0 =2,500. After each healing, its lifetime is decreased by 10, that is, $L_i - L_{i-1} = 10$ $(1 \leq i \leq 250)$. Based on Equation (1), we find that the total lifetime of TLC NAND flash memory is extended to $\sum_{i=0}^{250} L_i$ = (2500+2490+...+10) = 313,750 that is about 125 times longer than that of traditional TLC NAND flash memory.

2.3 The Architecture of Self-Healing NAND Flash Memory

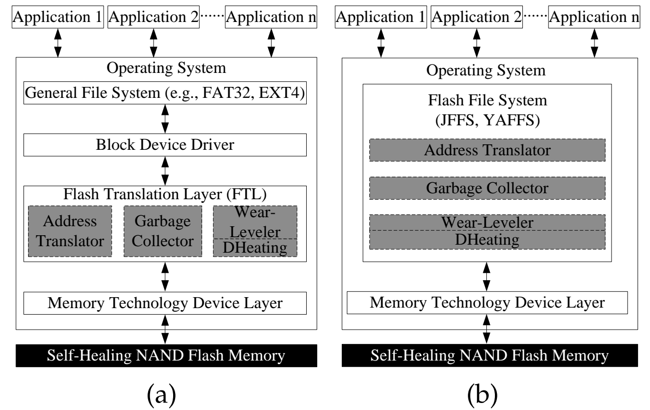


Fig. 5. (a) FTL-based self-healing NAND flash memory storage systems. (b) Flash-file-system-based self-healing NAND flash memory storage systems.

In general, there are two methods to make self-healing flash memory usable in embedded systems: *FTL-based flash memory storage systems* and *flash-file-system-based memory storage systems*. As shown in Fig. 5 (a) and Fig.5 (b), in these methods the MTD (Memory Technology Device) layer directly operates on a self-healing flash memory chip by providing primitive functions such as read, write, and erase operations; an FTL or a flash-file-system is used to manage NAND flash by handling such issues as out-of-place update, erase before rewrite, and limited lifetime. Both an FTL and a flash-file-system consist of three components:

an address translator, a garbage collector and a wear-leveler. The address translator translates addresses between the logical page number (LPN) and the physical page number (PPN). The garbage collector reclaims space by erasing obsolete blocks in which invalidated data exist; the wear-leveler is an optional component that distributes write or erase operations evenly across all blocks, so that the lifetime of a flash memory system can be improved.

In this paper, we mainly focus on solving the wear-leveling problem for self-healing NAND flash memory chips.

2.4 Wear Leveling

The principle in existing wear-leveling techniques is to evenly distribute writes and erases [23], [24], [25], [26], [27]. Basically, data are classified as hot or cold data according to data update frequency, while physical blocks are divided into old or young blocks. To evenly disperse erasures to all physical blocks, hot and cold data are allocated to young and old physical blocks, respectively. Existing techniques can be further divided into two categories: static and dynamic.

With the dynamic strategy, wear leveling is performed by allocating young blocks to hot data, while with the static strategy cold data are swapped with hot data so cold data cannot stay in young blocks for a long period of time [9], [10], [12]. In the dynamic wear leveling, if a young block is occupied by cold data but no update has occurred for a long period of time, it will not be recycled and other blocks will be worn out by updates of hot data. Therefore, static wear leveling is proposed to overcome the shortage of dynamic wear leveling.

SWL [12] is a typical static wear-leveling strategy. In this strategy, a BET (Block Erasing Table) is used to record which block has been erased in a predetermined time frame. In order to save memory space, a one-to-many mapping mode is adopted in SWL. That is, one BET flag can be shared by 2^k physical blocks. If one of the 2^k physical blocks is erased, the corresponding BET flag is set at 1. At the same time, two numbers, f_{cnt} and e_{cnt}, are used to record the number of 1 in the BET and the total number of block erases done since the BET was reset, respectively. When the ratio of f_{cnt} and e_{cnt} is equal to or larger than a threshold T, which means that some level of unevenness has occurred or many erases have been done on a small number of blocks, SWL will find out those used blocks whose corresponding BET flag is 0, and these blocks will be swapped with old blocks. SWL can effectively conduct wear leveling with a small space overhead. Therefore, in this paper, we compare our method with SWL.

2.5 Motivating Example

A motivating example is shown in Fig. 6. Assume that there are eight blocks and one block will carry out a heating operation after 2,500 erasures in a self-healing NAND flash memory chip.

As shown in Fig. 6(a), SWL [12] causes all of the blocks to be worn out evenly. As a result, at tn, all eight physical blocks reach their healing threshold of 2,500, and the concentrated heating problem occurs. This can exhaust the energy in the battery within a very short time period and reduce the lifetime of the battery.

Having made this observation, our idea in DHeating is to disperse the heating of the eight blocks over different times. As shown in Fig. 6(b), at $t2$, only Block 0 and Block 1 reach the heating bar. At t_m, after Block 6 and Block 7 complete heating operations, all blocks finish the heating operation.

Normally, $tm > tn$, since previous wear-leveling strategies have introduced some valid page copies and block erasure overheads to cause all blocks to wear out evenly. On the other hand, our proposed DHeating scheme only introduces these overheads when the system carry out a heating operation. As a result, our proposed scheme can further extend the lifetime of self-healing flash memory and improve the performance of the system.

3 The DHeating Scheme

In this section, we introduce the DHeating scheme to effectively address the concentrated heating problem. We first give an overview in Section 3.1, and then present the dispersed heating strategy and the lazy heating repair scheme in Sections 3.2 and 3.3, respectively. In Section 3.4, we discuss the early heating technique. An example to show how DHeating works with NFTL is presented in Section 3.5. Finally, we analyze the performance and overhead in Section 3.6.

3.1 Overview

The DHeating strategy consists of three schemes: dispersed heating, lazy heating repair and early heating. The dispersed heating scheme is to disperse heating operations to address the concentrated heating problem. The basic idea of the dispersed heating scheme is to intensively wear a small number of flash memory cells at a time. These cells will be worn out quickly and be heated much earlier than that of the other cells. Then, we swap out these heated flash memory cells, and repeat the above procedure. To this end, data are divided into two categories, hot data and cold data, based on data update frequency. According to block erasure times, physical blocks are classified into old, young, and new physical blocks. Old physical blocks are stored in the old pool, young physical blocks are kept in the young pool, and new physical blocks are stored in the new pool. When a write request is received, the dispersed heating scheme will check which pool the data request belongs to and allocate the corresponding physical blocks. In addition, the

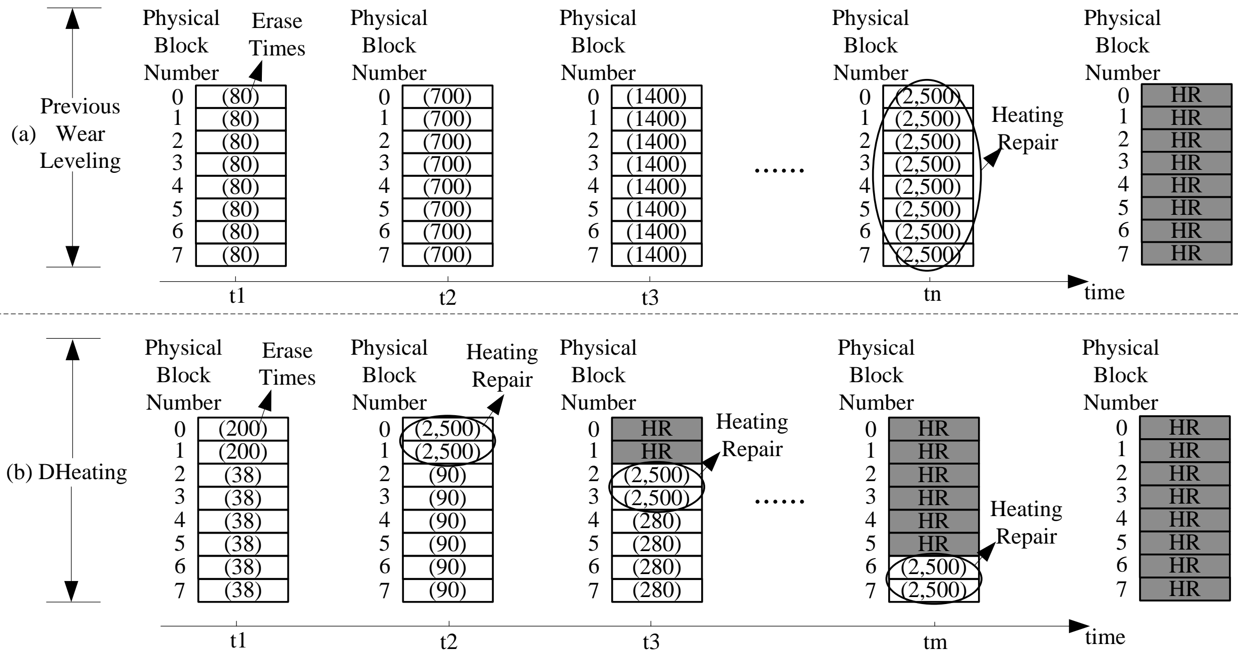


Fig. 6. Motivating example.

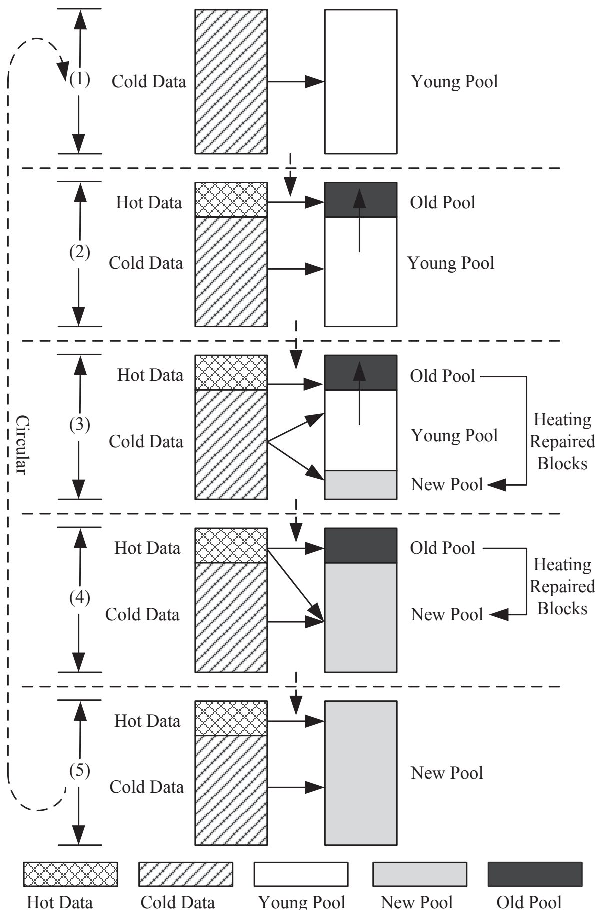


Fig. 7. Dispersed heating scheme.

lazy heating repair scheme is proposed to address the long time heating issue. The heating procedure takes several seconds and seriously degrades the I/O performance. In order to address this issue, the lazy heating repair scheme delays the heating operation and employs the system idle time to repair. Furthermore, the flash memory's reliability becomes worse with the flash memory cells reaching the excepted worn-out time. We propose an early heating strategy to solve the reliability problem. With the extended lifetime provided by self-healing, we can trade some lifetimes for reliability. The idea is to start the healing process earlier than the expected worn-out time. With DHeating, the energy consumed during each occurrence of heating is minimized, the long time heating effect can be eased, and the reliability of self-healing flash memory is enhanced.

3.2 Dispersed Heating

In the dispersed heating scheme, all blocks are categorized into three types, namely, young, old, and new blocks and are accordingly put into the young, old and new pools. Meanwhile, we divide the data into cold and hot and put them into blocks in different pools based on the different stages of our scheme.

The dispersed heating scheme has five stages, as shown in Fig. 7. Stage 1 is the starting point, where cold data are stored in young blocks. In Stage 2, we divide the hot and cold data and the young and old blocks. In Stage 3, old blocks will be heated and moved to the new pool. In Stage 4, no valid blocks exist in the young pool, and all blocks are kept in the old pool and the new pool. In Stage 5, all old blocks are healed and put into the new pool. We can then

repeat the above five stages for a new round. Next, we will present the details of each stage and introduce a key function used in our scheme, the *hot data filter*.

Stage 1 is the initial stage. In Stage 1, all data are cold data and all physical blocks are stored in the young pool.

In Stage 2, hot/cold data and young/old blocks are classified. Based on block erasure times, young blocks will be selected as old blocks and put in the old pool, since we do not want old blocks to be allocated to cold data and young blocks to be allocated to hot data. Then, the blocks in the old pool will continue to be used by the hot data. These blocks will be healed first. A *hot data filter* will be used to identify hot/cold data and young/old blocks, which will be discussed later.

In Stage 3, we start to heal blocks in the old pool, and these healed blocks are moved to the new pool. Then cold data are allocated to blocks in the new and young pools, while hot data continue to be assigned to blocks in the old pool. We move healed blocks to the new pool, because if they are kept in the old pool, they will be used by hot data and may reach their permanent retiring time much earlier than the other blocks. This can lead to very early shrinkage of the capacity of NAND flash memory. When allocating cold data, we will first use blocks in the young pool so blocks in the young pool will become old and move to the old pool, and blocks in the new pool will only be utilized until there are no free blocks in the young pool. In this way, the size of the young pool will decrease while that of the new pool will increase.

We enter Stage 4 after all of the blocks in the young pool are used up. At this stage, there are only the new and old pools, containing new and old blocks, respectively. Different from the above stages, in Stage 4 hot data can be allocated to blocks in both the old and new pools. However, blocks in the new pool are only used when there are no free blocks in the old pool. In the new pool, a free block with the smallest number of erasures is used to hold hot data, and this block will be kept in the new pool after it has been reclaimed. In the old pool, blocks will be healed and moved to the new pool As a result, all blocks will become new at some point in time, and we will enter Stage 5.

Stage 5 is the last stage, and only the new pool exists. The main function of this stage is to clear up history records such as update times and hot or cold data. Then, we will go back to Stage 1 and repeat the above procedures until a flash memory enters the last heating phase, and permanently retires.

Hot data filter: Hot/cold data are identified based on the update frequency. We use T to represent the number of hot data threshold that is calculated as follows:

$$T = (\sum_{i=0}^{n} U_i)/n \tag{2}$$

In this equation, n denotes the number of blocks updated by hot and cold data during a time period and U_i $(1 \leq i \leq n)$ is the update times of i's logical block. $U_{max}(= Max_{1 \leq i \leq n} U_i)$ denotes the maximum update times of all cold data. If U_{max} is equal to or larger than the threshold T, a new piece of hot data is detected. Correspondingly, the oldest block or the block with the largest number of erasures in the young pool will be moved to the old pool. The threshold T is calculated based on the average update times during a time period because of the following considerations:

- If T is selected larger than the average update times, some hot data may not be identified. For example, if all update requests are issued from one piece of data, and U_{max} is equal to the average update times, then, this piece of hot data cannot be identified.
- If T is selected smaller than the average update times, too many pieces of cold data may be selected as hot data. This will lead to the selection of many blocks as hot data.

For each logical block, we use a number to store its page update times. Initially, all numbers are set as zero. When one of the numbers achieves its maximum value, the *hot data filter* will be triggered, and all numbers will be reset as zero. It is important to determine how many bits are used to represent the number. If too many bits are used, a large memory space will be consumed and hot data will be detected for a very long period of time; otherwise, the *hot data filter* will be frequently triggered. In our experiments, eight bits are used to represent this number. An example is given below to show how hot data are identified. Assume that there are eight logical blocks, and eight bits are used to record the number of updates of each logical block. Suppose that the update times of these eight logical blocks are 255, 100, 3, 2, 6, 2, 8, and 8, respectively. When the *hot data filter* is triggered, 255 is the maximum value of an 8-bit unsigned number. Using Equation (2), the threshold T can be calculated as (255+100+3+2+6+2+8+8)/8 = 48. If the logical block with the update time 255 has already been chosen as hot data and all other blocks are cold data, then U_{max} is 100 as it is selected from all cold data. Because U_{max} is larger than 48, the *hot data filter* chooses the logical block with the update time 100 as a new piece of hot data.

Algorithm 1 describes how the *hot data filter* works. Three input parameters are used in *hot data filter*: n, U_{all} and U_{max}, which denote the number of pieces of updated data, all update times, and the maximum update times of cold data respectively. n and U_{all} are used to calculate filter threshold T. Then if U_{max} is equal to or larger than the threshold T, a new piece of hot data will be detected and returned, and a physical block with the maximum erase time from the young pool will be moved to the old pool. Otherwise, if U_{max}

Algorithm 1: Hot data filter

Input: n: The number of updated blocks
U_{all}: All update times
U_{max}: Maximum update times of cold data
E: The erasure times of physical blocks
P_y: The young pool
P_o: The old pool
Output: HDN: The logical block number identified as the hot data

1 $T \leftarrow U_{all}/n$
2 **if** $U_{max} \geq T$ **then**
3 $HDN \leftarrow$The logical block number with U_{max}
4 $PB_{oldest} \leftarrow E_{max\{X|X \epsilon P_y\}}$
5 $P_o \leftarrow PB_{oldest}$
6 Return HDN
7 **else**
8 Return NULL to denote no hot data is detected

is smaller than threshold T, no hot data is detected and the algorithm returns $NULL$.

3.3 Lazy Heating Repair

The heating operation takes a few seconds, which are much longer than that of an erase operation [22], such as $1.5ms$ block erase time [2]. In addition, a heating operation can block the whole Die response in a flash chip and thus may seriously degrade the system response time. Therefore, the heating operation has become the new performance bottleneck in self-healing flash memory and an effective management method is urgently required. In this section, we propose a lazy heating repair technique to alleviate the long heating effect with the benefits of utilizing system idle time.

When a block reaches the heating point, the block is moved to a heating list instead of being heated immediately. Then, when the system is idle, the blocks in the heating list are selected to repair. By delaying the heating operation and employing the idle time to repair, the lazy heating repair technique can ease the long time heating effect. However, this strategy may introduce a new problem. If too many blocks are accumulated in the heating list, these blocks may be healed in a concentrated manner. In order to address this issue, the lazy heating repair is periodically triggered to clean the heating list. The heating period is adaptively adjusted according to two factors: the number of heating blocks in the heating list and the number of free blocks. If a lot of blocks are waiting to do heating repair in the heating list and a few free blocks are available, the heating repair operation is triggered intensively within a short heating period. On the other hand, a few blocks in the heating list or many free blocks will result in triggering the heating operation infrequently.

Fig. 8 illustrates the procedure of the lazy heating repair scheme. Assume that the heating period is $\triangle t$

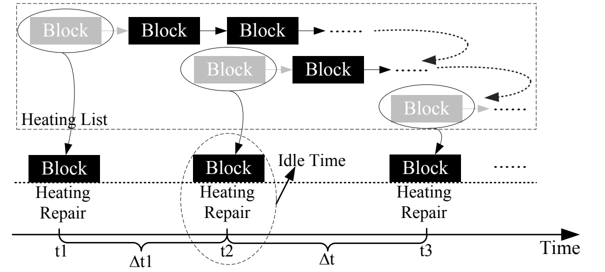


Fig. 8. Illustration on the procedure of lazy heating repair.

and a block is heated at $t1$. If the system is idle after $\triangle t1 (\triangle t1 < \triangle t)$ time interval, a block in the heating list will be selected to repair at the idle time, such as $t2$ as shown in Fig. 8. On the other hand, if the system is always busy during the $\triangle t$ time interval, a block in the heating list is chosen to repair after the $\triangle t$ time interval. As shown in Fig. 8, the time interval between $t2$ and $t3$ is $\triangle t$ and the system is always busy during this time interval. Then, at $t3$, a block in the heating list is selected to repair, since the elapsed time has reached the heating period $\triangle t$.

3.4 Early Heating

An early heating strategy is proposed to enhance the reliability for self-healing NAND flash memory. In the dispersed heating scheme, where write and erase operations are on a small portion of flash memory cells, the reliability problem might arise. We propose to utilize an early heating strategy to address this problem. The idea is to start the healing process earlier than the expected worn-out time.

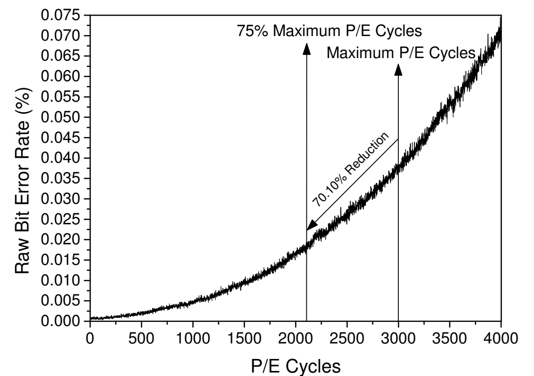


Fig. 9. The raw bit error rate with different P/E cycles [15].

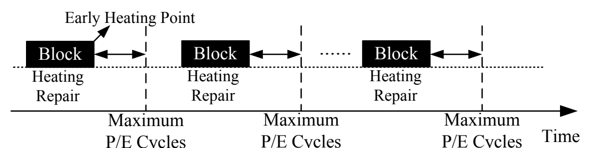


Fig. 10. The early heating strategy for reliability.

Since the dispersed heating scheme utilizes a small number of flash memory cells in a concentrated manner, the reliability problem may arise. Fig. 9 shows the raw bit error rate with different P/E cycles. These results are obtained via testing Micron's MLC flash 29F64GCBAAA with the capacity of 64 Gb and the maximum of 3000 P/E cycles. One key observation form Fig. 9 is that the raw bit error rate increases dramatically with flash memory cells reaching the expected maximum P/E cycles. If we do healing earlier than the expected maximum P/E cycles as shown in Fig. 10, the raw bit error rate can be greatly reduced and thus the reliability of self-healing flash memory can be effectively enhanced. For example, the bit error rate can be reduced by 70.10% with the cost of 25% lifetime reduction.

3.5 DHeating Working with NFTL

Fig. 11 illustrates how the DHeating scheme works. For purpose of demonstration, NFTL [12] is selected as the FTL. NFTL uses a block-level address translation mechanism for coarse-grained address translation and is widely used in embedded systems. Note that our scheme is general and can work with other FTLs at block-level, page-level, or hybrid-level. In NFTL, a logical page number (LPN) is divided by the number of pages in a block to obtain its logical block number (LBN) and block offset, where the LBN is the quotient, and the block offset is the remainder of the division. A block-level mapping table maps the LBN into a physical block known as the primary block (PPBN). Each primary block is associated with some additional physical blocks known as replacement blocks (RPBN). A write operation to an LPN is mapped to a page in a primary block, and subsequent update operations to the same LPN are made on the corresponding replacement block with the same block offset.

Fig. 11(a) shows the first stage of the dispersed heating, where all data update times are 0 and all physical blocks are young. After some data requests are issued, hot data LBN 0 is filtered from cold data with the help of a *hot data filter* (Fig. 11(b)). Correspondingly, two old blocks (block 0 and block 1) with the erasure times of 210 and 200 respectively are stored in the old pool.

Fig. 11(c) shows that block 0 and block 1 have been healed and moved from the old pool to the new pool. Moreover, two old blocks (block 2 and block 3) are moved from old pool to new pool. In Fig. 11(d), when all the young blocks have been depleted; and only old blocks and new blocks are stored in the old pool and new pool respectively, we enter Stage 4. After the blocks in the old pool have healed, we enter the last stage as shown Fig. 11(e).

3.6 Performance and Overhead Analysis

In this section, we analyze the system performance and overhead of DHeating by comparing it with the SWL scheme[12]. The system response time is an important metric to evaluate the performance of FTLs. It is the time period from the point when an operation is issued to the point when the operation has been completed. The inputs of an FTL are read and write operations. We conduct the analysis for the system response time of read and write operations. The symbols used in this analysis are listed below.

T_{rd}	The time to read one page
T_{wr}	The time to write one page
T_{erase}	The time to erase a block
N_{vpage}	The number of valid pages in one block

The performance overhead is reflected as extra valid page copies and block erasures, and the time is:

$$(T_{rd} + T_{wr}) \times N_{vpage} + T_{erase}. \quad (3)$$

Compared with SWL, DHeating only introduces small performance overhead. In SWL, cold data and hot data are swapped very frequently so as to achieve static wear leveling. The frequent swap operations result in heavy performance overhead. SWL will be triggered to find old and young blocks, and swap valid data between the two different kinds of blocks when unevenness occurs. On the other hand, DHeating only does a data swap when young blocks become old or old blocks become new. Therefore, compared with the SWL scheme, DHeating can reduce valid page copies and block erasures, thereby improving the system response time.

DHeating requires more memory space than that of SWL to record the update times of each block. However, the memory space overhead is negligible since the information recorded is at the block level. For example, the block size of a 32Gb MLC NAND flash [2] memory chip is 512KB. If we use 2 bytes to record the erasure times for each block, only 16KB memory space is required.

With the early heating scheme, DHeating trades lifetime for reliability. However, as shown in the experiments, with only 5% of lifetime overhead, heating can still be dispersed very well. Indeed, the reliability is also improved in DHeating.

4 EVALUATION

In this section, we present our experimental results with analysis. We compare and evaluate the proposed DHeating scheme with the baseline scheme [12] in terms of four metrics: the consecutive heating time intervals, the extra valid page copies, the lazy heating effect and the early heating effect. The performance evaluation is conducted on an embedded development board with a Samsung ARM11 processor and a 8 Gb NAND flash memory chip.

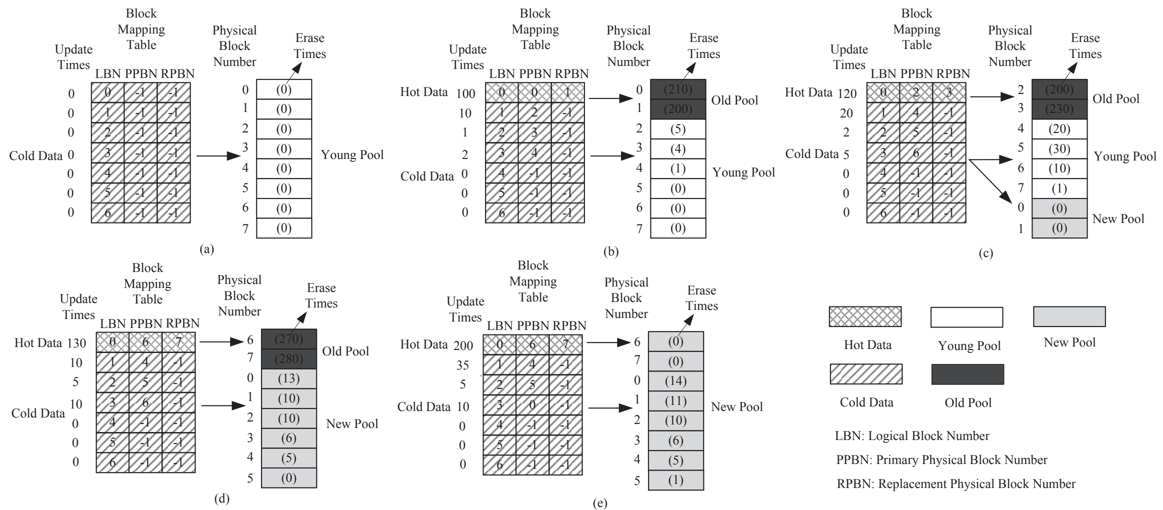


Fig. 11. Illustration of the dispersed heating scheme working with NFTL [12].

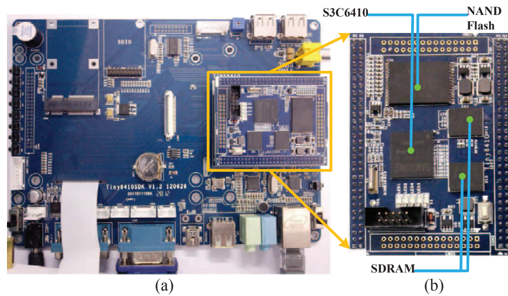


Fig. 12. Experimental platform. (a) The top layer of our experimental platform. (b) The core development board.

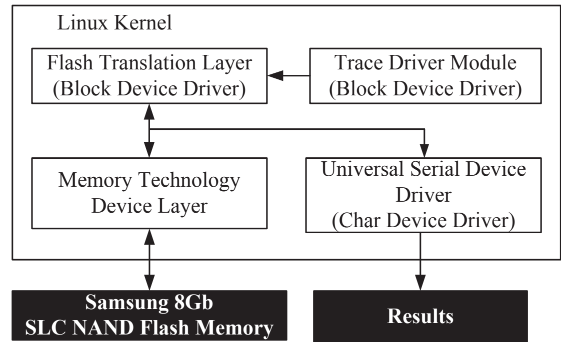


Fig. 13. The evaluation framework of DHeating.

4.1 Experimental Setup

We conducted experiments on a hardware platform. Fig. 12(a) shows the top view of our hardware platform. The evaluation platform adopts an ARM11 processor core (Samsung S3C6410 [28]) with ARMv6 architecture. In this platform, the ARM processor core runs at 532MHz; and consists of a 16 KB instruction cache and a 16 KB data cache. The platform adopts the Linux kernel 2.6.38. The core board is equipped with 8 Gb of NAND flash memory and 256 MB of SDRAM. The physical interfaces, such as the RJ45 interface, are designed in the mother board. One pin connector is used to connect the core board with the mother board.

The evaluation framework of our work is shown in Fig. 13. DHeating is implemented as a block device driver in the Linux kernel 3.5. DHeating functionally works as the wear-lever of the flash translation layer (FTL), and another trace driver module is implemented to trigger DHeating. In our evaluation, the trace driver module is also implemented as a block device driver. FTL can issue read/write operations to the memory technology device (MTD) layer, which can control the NAND flash memory chip. We utilize the universal serial device driver (i.e. a char device driver) to obtain and output the experimental results.

For fair comparisons, the same configuration have been adopted for both the baseline scheme and the proposed DHeating scheme. We emulate the process of self-healing and assume that NAND flash memory will perform heating after a given lifetime.

We use real applications as benchmarks to evaluate the effectiveness of DHeating. Since the real environment varies significantly and the same application may generate different I/O requests with different running time even with the same configuration, we collect the I/O requests of real applications and use the collected traces to evaluate the techniques in order to make a fair comparison. The characteristics of traces are shown in Table 1. These applications are typical operations in our daily lifetime. They are mainly write dominant, and they can be used to accelerate the evaluation process and evaluate the performance of worn-out flash memory cells for NAND flash memory. We test each benchmark on the evaluation platform. The traces iteratively issue requests to the storage system.

4.2 Results and Discussion

In this section, we present the experimental results with analysis. We first present the heating impact of the self-healing flash memory. Then, we present

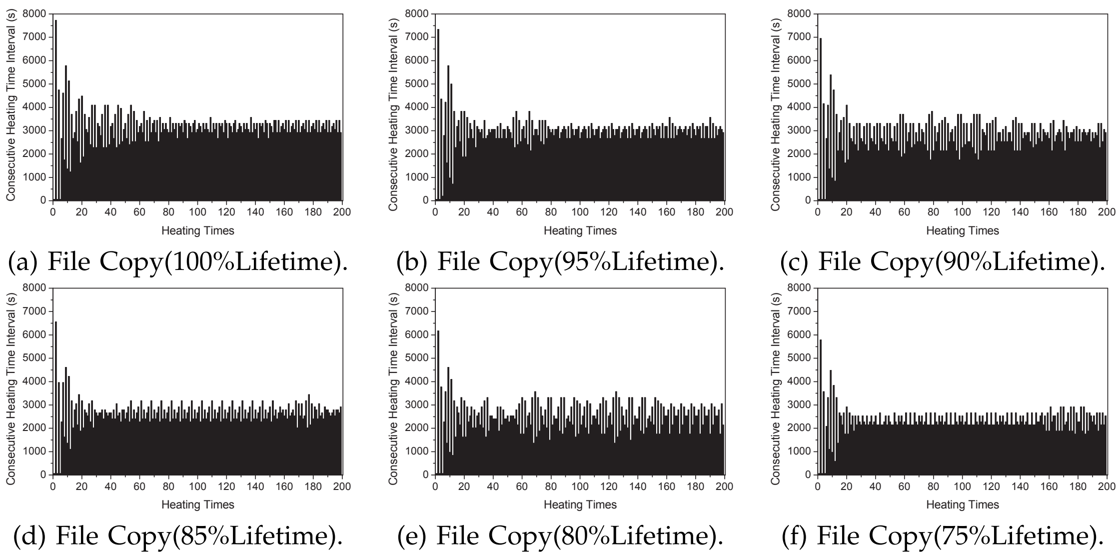
(a) SWL(File Copy). (b) SWL(Sensor). (c) SWL(Multimedia).

(d) DHeating(File Copy). (e) DHeating(Sensor). (f) DHeating(Multimedia).

(g) SWL(NFS). (h) SWL(SD Card). (i) SWL(FTP).

(j) DHeating(NFS). (k) DHeating(SD Card). (l) DHeating(FTP).

Fig. 14. The consecutive heating time interval of SWL and DHeating over six applications.



(a) File Copy(100%Lifetime). (b) File Copy(95%Lifetime). (c) File Copy(90%Lifetime).

(d) File Copy(85%Lifetime). (e) File Copy(80%Lifetime). (f) File Copy(75%Lifetime).

Fig. 15. The consecutive heating time intervals of DHeating over $File\ Copy$ application with different lifetimes.

Benchmarks	Numbers of Request	Write (%)	Avg.Arr. Time (ms)	Avg.Req. Size (KB)
File Copy	645,895	90.83	8.19	3.17
Sensor	777,945	92.66	8.12	2.26
Multimedia	790,925	83.10	196.90	48.03
NFS	850,700	89.37	8.40	29.38
SD Card	987,970	94.95	38.30	49.87
FTP	518,575	97.4	72.97	20.05

TABLE 1
The characteristics of the applications.

the improvement in performance by comparison of DHeating scheme and the baseline scheme. Finally, we analyze the early heating effect.

We use SWL and DHeating to represent the results obtained from the work in [12] and the proposed DHeating scheme, respectively.

4.2.1 Heating Dispersal

Benchmarks	SWL Ave. Heating Time Int.(s)	DHeating Ave.Heating Time Int.(s)	DHeating/SWL
File Copy	162.663	3,143.191	19.323
Sensor	159.070	3,789.925	23.825
Multimedia	166.572	3,864.397	23.199
NFS	185.844	4,152.161	22.342
SD Card	176.371	4,831.884	27.395
FTP	169.512	2,514.422	14.833

TABLE 2
Average heating time interval.

Table 2 shows the average consecutive heating intervals of two physical blocks. The consecutive heating interval denotes the frequency of heating. Therefore, the longer the consecutive heating interval that the NAND flash memory experiences, the better its performance and lifetime. Judging from the experimental results, our DHeating scheme can delay heating by up to 24 times compared to SWL. This shows the effectiveness of DHeating in maximizing the consecutive heating interval.

Fig. 14 presents the experimental results of the consecutive heating time intervals of SWL and DHeating. From the results, we can see that the consecutive heating time interval for SWL is very short, and that our scheme can delay the consecutive heating time interval for much longer thant SWL can. In the evaluation, we assume that self-healing flash memory will trigger heating for every 100 program/erase (P/E) cycles.

Taking the benchmark *sensor* as an example, SWL finishes the 200th block heating in $31,655$ seconds, while our scheme finishes the 200th heating in $754,195$ seconds , which is 23 times longer. As the lifetime increases, DHeating will also significantly increase in improvement over SWL.

4.2.2 Performance Improvement

Previous wear-leveling schemes basically relied on the swapping of hot and cold data to balance the erase counts [11], [9], [10]; therefore, some system performance had to be sacrificed, such as extra valid page copies and extra block erasures. In our technique, we allocate hot data to old blocks and simply swap the data when young blocks become old blocks or old blocks become new blocks. Therefore, DHeating can incur much less extra overhead compared to previous wear-leveling schemes.

Benchmarks	SWL	DHeating	DHeating over SWL (%)
File Copy	158,720	9,344	94.11
Sensor	209,920	10,496	95.00
Multimedia	158,720	7,936	95.00
NFS	206,036	9,472	95.40
SD Card	215,096	8,640	95.98
FTP	139,202	10,688	92.32

TABLE 3
Extra valid page copies.

Benchmarks	SWL	DHeating	DHeating over SWL (%)
File Copy	3,262	192	94. 11
Sensor	3,800	190	95.00
Multimedia	3,600	180	95.00
NFS	4,010	184	95.41
SD Card	4,508	181	95.99
FTP	2,394	183	92.36

TABLE 4
Extra block erasures.

The extra valid page copies and the extra number of block erase counts are compared in Table 3 and Table 4, respectively. The results of the experiment show that DHeating performs significantly better than SWL in reducing the extra valid page copies and the extra number of block erase counts. Since the extra number of valid page copy operations will issue more write operations to blocks containing free pages, more erase operations will be incurred and the lifetime of the NAND flash memory will be shortened. DHeating can effectively reduce the number of extra valid page copy operations and the extra number of block erase counts, which are beneficial to a NAND flash memory storage system.

4.2.3 Lazy Heating Repair Effect

Lazy heating repair is proposed to eliminate the long heating time effect. In this experiment, the heating time is configured to $3s$. In order to study the heating period effect, the heating period is configured to $60s$ and $600s$, respectively. The system idle time is detected based on the pending requests. If the pending request queue is empty, the system is idle.

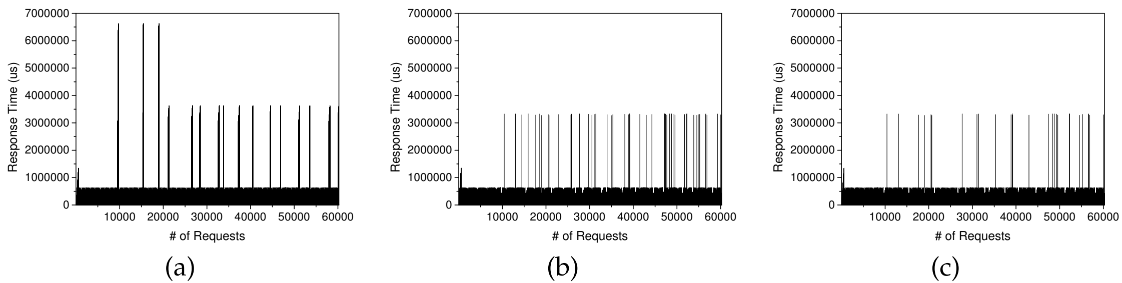


Fig. 16. The lazy heating repair effect with running the $File\ Copy$ application. (a) The baseline scheme without lazy heating repair. (b) The lazy heating repair technique with $60s$ heating period. (c) The lazy heating repair technique with $600s$ heating period.

Fig. 16 illustrates the lazy heating repair effect. The dispersed heating scheme without the lazy heating repair technique is adopted as the baseline scheme. In the baseline scheme, the self-healing flash memory heats the worn-out cells immediately. As a result, the response time is greatly degraded as shown in Fig. 16 (a), such as $6632964us$ in the worst case. By employing the lazy heating scheme, the system response time can be effectively improved with the benefits of utilizing the system idle time as shown in Fig. 16 (b) and Fig. 16 (c). The benefits of the lazy heating repair scheme are reflected into two aspects, the worst case response time and the total heating overhead. Compared with the baseline scheme, the worst case response time is reduced by 49.77% and the total heating time overhead is improved by 98.84% on average. In addition, with the heating period prolonged, such as from $60s$ to $600s$, the lazy heating repair scheme works better and the improvement is enhanced by 43.49%. However, this improvement benefits from keeping more worn-out blocks in the heating list. If only a few free blocks are available to response the write requests and a lot of blocks are kept in the heating list, the blocks in the heating list may be required to be healed in a concentrated manner. This may seriously degrade the system response time. In order to well utilize the lazy heating repair scheme, it is better to dynamically adjust the heating period according to the number of heating blocks in the heating list and the number of free blocks.

4.2.4 Early Heating for Reliability

Early heating strategy can enhance the reliability at the expense of trading some flash memory lifetimes. With heating repair started earlier than the expected maximum P/E cycles, the reliability of flash memory can be effectively enhanced. For example, the bit error rate can be reduced by 70.10% with the cost of 25% lifetime reduction [15].

Table 5 shows the experimental results for early heating with different lifetimes. It can be seen that, using 95% as the heating threshold, early heating can work very well and the average heating time interval only decreases by an average of about 5%. Fig. 15 illustrates how different threshold values influence the consecutive heating time intervals of the application *File Copy*. With the heating threshold decrease, the average heating interval decreases accordingly. However, the proposed scheme can still work well with the decrease in the lifetime of the flash memory, and the reliability of the self-healing flash memory is enhanced.

Benchmarks	DHeating 100% Life Time Ave. Heat. Time(s)	DHeating 95% Life Time Ave. Heat. Time(s)	Ave. Early Heat. Diff. (s)
File Copy	3,143.191	2981.181	162.010
Sensor	3,789.924	3,596.558	193.366
Multimedia	3,864.396	3,663.518	200.878
NFS	4,152.160	3,931.683	220.477
SD Card	4,831.884	4,569.271	262.613
FTP	2,514.422	2,383.442	130.980

TABLE 5
A comparison of early heating over different applications with different reduced lifetimes.

5 CONCLUSION

In this paper, we proposed a scheme, called DHeating, to solve the concentrated heating problem and overcome the constrains of self-healing NAND flash memory. DHeating consists of three techniques: the dispersed heating technique, the lazy heating repair technique, and the early heating technique. These three techniques are proposed based on different considerations and interact with each other. The dispersed heating technique is designed to avoid the concentrated heating problem. The lazy heating repair technique can address the long time heating issue and the early heating technique is proposed to enhance the reliability of self-healing flash memory. We conducted experiments on a set of representative I/O workloads collected from the embedded development board. The experimental results show that our proposed scheme not only solves the concentrated heating problem for NAND flash memory, but also improves the system response time and enhances the reliability of the self-healing flash memory.

6 Acknowledgments

The work described in this paper is partially supported by the grants from Special Administrative Region, China (GRF 152138/14E), National Natural Science Foundation of China (Project 61272103, 61373049 and 61309004), National 863 Program 2013AA013202, Research Fund for the Doctoral Program of Higher Education of China (20130191120030), Chongqing cstc2012ggC40005 and cstc2013jcyjA40025, Fundamental Research Funds for the Central Universities (CDJZR14185501) and Chongqing University (2012T0006), and the Hong Kong Polytechnic University (4-ZZD7,G-YK24, G-YM10 and G-YN36). A preliminary version of this work appears in the Proceedings of IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2013) [29].

References

[1] C. Mellor, "TLC flash gets tender loving care from DensBits," *http://www.theregister.co.uk/2012/05/02/densbit_tlc/*, 2012.

[2] S. Electronics, "K9LBG08U0M(v1.0)-32GB DDP MLC," *http://www.samsung.com*.

[3] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang, "SDF: Software-defined flash for web-scale internet storage systems," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, 2014, pp. 471–484.

[4] J. Guo, J. Yang, Y. Zhang, and Y. Chen, "Low cost power failure protection for MLC NAND flash storage systems with PRAM/DRAM hybrid buffer," in *Design, Automation Test in Europe Conference Exhibition (DATE '13)*, 2013, pp. 859–864.

[5] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10)*, 2010, pp. 1–14.

[6] P. Desnoyers, "Analytic models of ssd write performance," *ACM Transactions on Storage (TOS)*, vol. 10, no. 2, pp. 8:1–8:25, 2014.

[7] 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 IEEE International Electron Devices Meeting (IEDM '12)*, 2012, pp. 9.1.1–9.1.4.

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

[9] L.-P. Chang and L.-C. Huang, "A low-cost wear-leveling algorithm for block-mapping solid-state disks," in *Proceedings of the 2011 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '11)*, 2011, pp. 31–40.

[10] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*, 2007, pp. 1126–1130.

[11] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," *IEEE Transactions on Computers*, vol. 59, pp. 53 –65, 2010.

[12] ——, "Endurance enhancement of flash-memory storage, systems: An efficient static wear leveling design," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, 2007, pp. 212 –217.

[13] C. Wang and W.-F. Wong, "Observational wear leveling: An efficient algorithm for flash memory management," in *Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference (DAC '12)*, June 2012, pp. 235–242.

[14] T.-W. Kuo, Y.-H. Chang, P.-C. Huang, and C.-W. Chang, "Special issues in flash," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '08)*, 2008, pp. 821–826.

[15] M. Huang, Z. Liu, and L. Qiao, "Asymmetric programming: A highly reliable metadata allocation strategy for MLC NAND flash memory-based sensor systems," *Sensors*, vol. 14, no. 10, pp. 18 851–18 877, 2014.

[16] Y. Cai, E. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *Design, Automation Test in Europe Conference Exhibition (DATE '12)*, 2012, pp. 521–526.

[17] C. Gao, L. Shi, K. Wu, C. Xue, and E.-M. Sha, "Exploit asymmetric error rates of cell states to improve the performance of flash memory storage systems," in *2014 32nd IEEE International Conference on Computer Design (ICCD '14)*, 2014, pp. 202–207.

[18] G. Sun, X. Wu, and Y. Xie, "Exploration of 3D stacked L2 cache design for high performance and efficient thermal control," in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '09)*, 2009.

[19] W. Zhang, N. K. Jha, and L. Shang, "Low-power 3D NANO/CMOS hybrid dynamically reconfigurable architecture," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 6, no. 3, pp. 10.1–10.32, 2010.

[20] Y. Wang, Y. Liu, Y. Liu, D. Zhang, S. Li, B. Sai, M. Chiang, and H. Yang, "A compression-based area-efficient recovery architecture for nonvolatile processors," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE '12)*, 2012, pp. 1519–1524.

[21] W. Qi, G. Dong, and T. Zhang, "Exploiting heat-accelerated flash memory wear-out recovery to enable self-healing SSDs," in *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '11)*, 2011, pp. 1–5.

[22] 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 (DAC '14)*, 2014, pp. 83:1–83:6.

[23] 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 (CASES '07)*, 2007, pp. 160–164.

[24] L. Shi, J. Li, Q. Li, C. Xue, C. Yang, and X. Zhou, "A unified write buffer cache management scheme for flash memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2779–2792, 2014.

[25] M. Jung, W. Choi, J. Shalf, and M. T. Kandemir, "Triple-A: A non-SSD based autonomic all-flash array for high performance storage systems," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, 2014, pp. 441–454.

[26] P. Desnoyers, "Analytic modeling of SSD write performance," in *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR '12)*, 2012, pp. 12:1–12:10.

[27] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen, "FlexLevel: A novel NAND flash storage system design for LDPC latency reduction," in *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*, 2015, pp. 194:1–194:6.

[28] Samsung, "S3C6410," *http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7115&iaId=835*.

[29] R. Chen, Y. Wang, and Z. Shao, "DHeating: Dispersed heating repair for self-healing NAND flash memory," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '13)*, 2013, pp. 7:1–7:10.