ERROR ANALYSIS AND RETENTION-AWARE ERROR MANAGEMENT FOR NAND FLASH MEMORY

Contributors

Yu Cai
Carnegie Mellon University

Gulay Yalcin
Barcelona Supercomputing Center

Onur Mutlu
Carnegie Mellon University

Erich F. Haratsch
LSI Corporation

Adrian Cristal
Barcelona Supercomputing Center

Osman S. Unsal
Barcelona Supercomputing Center

Ken Mai
Carnegie Mellon University

With continued scaling of NAND flash memory process technology and multiple bits programmed per cell, NAND flash reliability and endurance are degrading. In our research, we experimentally measure, characterize, analyze, and model error patterns in nanoscale flash memories. Based on the understanding developed using real flash memory chips, we design techniques for more efficient and effective error management than traditionally used costly error correction codes.

In this article, we summarize our major error characterization results and mitigation techniques for NAND flash memory. We first provide a characterization of errors that occur in 30- to 40-nm flash memories, showing that retention errors, caused due to flash cells leaking charge over time, are the dominant source of errors. Second, we describe retention-aware error management techniques that aim to mitigate retention errors. The key idea is to periodically read, correct, and reprogram (in-place) or remap the stored data before it accumulates more retention errors than can be corrected by simple ECC. Third, we briefly touch upon our recent work that characterizes the distribution of the threshold voltages across different cells in a modern 20- to 24-nm flash memory, with the hope that such a characterization can enable the design of more effective and efficient error correction mechanisms to combat threshold voltage distortions that cause various errors. We conclude with a brief description of our ongoing related work in combating scaling challenges of both NAND flash memory and DRAM memory.

"...we summarize our major error characterization results and mitigation techniques for NAND flash memory."

Introduction

During the past decade, the capacity of NAND flash memory has increased more than 1000 times as a result of aggressive process scaling and multilevel cell (MLC) technology. This continuous capacity increase has made flash economically viable for a wide variety of applications, ranging from consumer electronics to primary data storage systems. However, as flash density increases, NAND flash memory cells are more subject to various device and circuit level noise, leading to decreasing reliability and endurance. The P/E cycle endurance of MLC NAND flash memory has dropped from ~10K for 5x-nm (that is, 50- to 59-nm) flash to around ~3K for current 2x-nm (that is, 20- to 29-nm) flash.[1][5] The reliability and endurance are expected to continue to decrease when 1) more than two bits are programmed per cell, and 2) flash cells scale beyond the 20-nm technology generations. This trend is forcing flash memory designers to apply even stronger error correction codes (ECC) to tolerate the increasing error rates, which comes at the cost of additional complexity and overhead.[4]

"...as flash density increases, NAND flash memory cells are more subject to various device and circuit level noise,..."

In our research at Carnegie Mellon University, we aim to develop new techniques that overcome reliability and endurance challenges of flash memory to enable its scaling beyond the 20-nm technology generations. To this end, we experimentally measure, characterize, analyze, and model error patterns that occur in existing flash chips, using an experimental flash memory testing and characterization platform we have developed.[2] Based on the understanding we develop from our experiments, we aim to develop error management techniques that aim to mitigate the fundamental types of errors that are likely to increase as flash memory scales. Our goal is to design techniques that are more effective and more efficient than stronger error correction codes (ECCs), which has been the traditional way of improving endurance and reliability of flash memory. In this article, we provide an overview of the results of our recent error characterization experiments[3][6] and describe some error mitigation techniques.[4]

"Our goal is to design techniques that are more effective and more efficient than stronger error correction codes (ECCs), which has been the traditional way of improving endurance and reliability of flash memory."

In particular, we have recently experimentally characterized complex flash errors that occur at 30- to 40-nm flash technologies[3], categorizing them into four types: retention errors, program interference errors, read errors, and erase errors. Our characterization shows the relationship between various types of errors and demonstrates empirically using real 3x-nm flash chips that retention errors are the most dominant error type. Our results demonstrate that different flash errors have distinct patterns: retention errors and program interference errors are program/erase-(P/E)-cycle-dependent, memory-location-dependent, and data-value-dependent. Since the observed error patterns are due to fundamental circuit and device behavior inherent in flash memory, we expect our observations and error patterns to also hold in flash memories beyond 30-nm technology node.

"Our characterization shows the relationship between various types of errors and demonstrates empirically using real 3x-nm flash chips that retention errors are the most dominant error type."

Based on our experimental characterization results that show that the retention errors are the most dominant errors, we have developed a suite of techniques to mitigate the effects of such errors, called Flash Correct-and-Refresh (FCR).[4] The key idea is to periodically read each page in flash memory, correct its errors using simple ECC, and either remap (copy/move) the page to a different location or reprogram it in its original location by recharging the floating gates before the page accumulates more errors than can be corrected with simple ECC. Our simulation experiments using real I/O workload traces from a variety of file system, database, and search applications show that FCR can provide 46x flash memory lifetime improvement at only 1.5 percent energy overhead, with no additional hardware cost.

Finally, we also briefly describe major recent results of our measurement and characterization of the threshold voltage distribution of different logical states in MLC NAND flash memory.[6] Our data shows that the threshold voltage distribution of flash cells that store the same value can be approximated, with reasonable accuracy, as a Gaussian distribution. The threshold voltage distribution of flash cells that store the same value gets distorted as the number of P/E cycles increases, causing threshold voltages of cells storing different values to overlap with each other, which can lead to the

"...we hope that the characterization, understanding, models, and mechanisms provided in this work (and in our aforementioned previous works) would enable the design of new and more effective error tolerance mechanisms that can make use of the observed characteristics and the developed models."

incorrect reading of values of some cells as flash cells accumulate P/E cycles. We find that this distortion can be accurately modeled and predicted as an exponential function of the P/E cycles, with more than 95-percent accuracy. Such predictive models can aid the design of more sophisticated error correction methods, such as LDPC codes[7], which are likely needed for reliable operation of future flash memories. Even though we will not describe these models in detail in this article, the interested reader can refer to Cai et al.[6] for more detail.

As flash memory continues to scale to smaller feature sizes, we hope that the characterization, understanding, models, and mechanisms provided in this work (and in our aforementioned previous works[3][4][6]) would enable the design of new and more effective error tolerance mechanisms that can make use of the observed characteristics and the developed models.

Flash Memory Background

NAND flash memory can be of two types: single level cell (SLC) flash and multilevel cell (MLC) flash. Only one bit of information can be stored in an SLC flash cell, while multiple bits (2 to 4 bits) can be stored in an MLC flash cell.[8][9][10] MLC flash represents n bits by using 2^n non-overlapping threshold voltage (V_{th}) windows. The threshold voltage of a given cell is mainly affected by the number of electrons trapped on the floating gate. Figure 1 shows the bit mapping to V_{th} and the relative proportion of electrons on the floating gates of a 2-bit MLC flash.

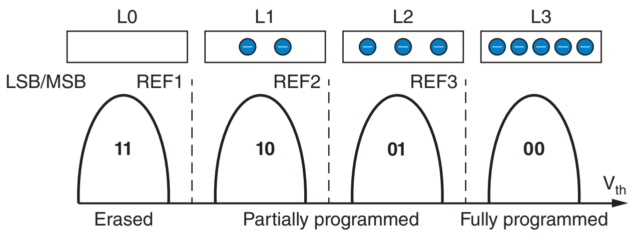


Figure 1: Threshold voltage distribution example of 2-bit MLC flash
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

A NAND flash memory chip is composed of thousands of blocks. Each block is a storage array of floating gate transistors. A flash block usually has 32 to 64 wordlines. The cells on the same wordline can be divided into two groups: even and odd, depending on the physical location. For SLC flash, each group corresponds to just one logical page, that is, even pages and odd pages. As an MLC flash cell stores multiple bits, the bits corresponding to the same logical location of a cell in a group form one logical page. For example, all the most significant bits (MSBs) of the cells of an even group form one MSB-even page. Similarly, other types of pages are MSB-odd page, LSB-even page, and LSB-odd page. The page number assignments for each bit of the flash memory are shown in Figure 2(a), ranging from 0 to 127 for the selected flash in this article. The size of each page is generally between 2 KB and 8 KB (16k and 64k bitlines). The stack of flash cells in the bitline direction forms one string. The string is connected to a bit line through SGD (the select gate at the drain end) and connect to the common source diffusion through SGS (the select gate at the source end) as shown in Figure 2(b). Flash memories generally support three fundamental operations as follows:

"Flash memories generally support three fundamental operations..."

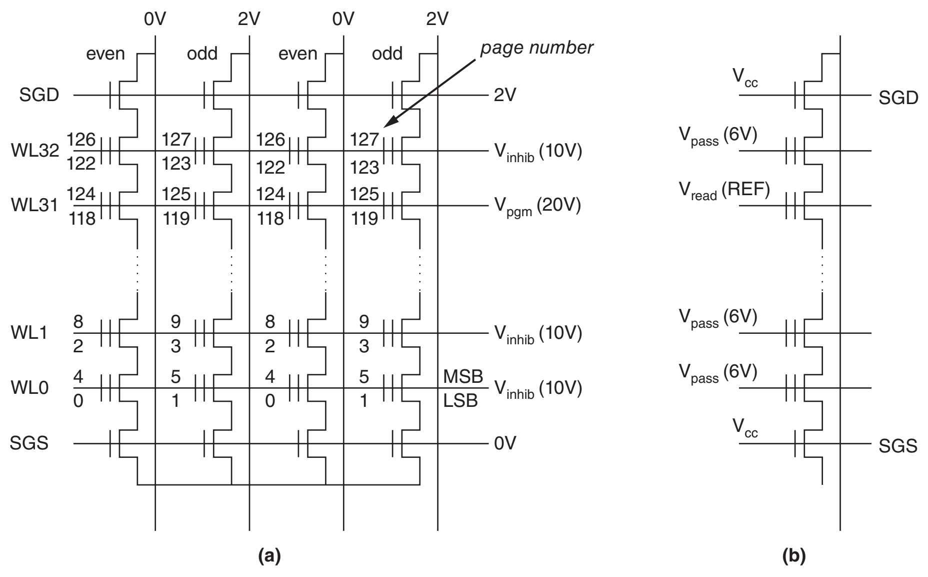


Figure 2: NAND flash organization and operations: (a) Partial block organization and program operation on page 118; (b) Read operation
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

Erase

During the *erase* operation, a high positive erase voltage (for example, 20V) is applied to the substrate of all the cells of the selected block and the electrons stored on the floating gate are tunnelled out through Fowler-Nordheim (FN) mechanisms.[9] After a successful erase operation, all charge on the floating gates is removed and all the cells are configured to the L0 (11) state. The erase operation is at the granularity of one block.

Program

During the *program* operation, a high positive voltage is applied to the wordline, where the page to be programmed is located. The other pages sharing the same wordline are inhibited (from being programmed) by applying 2V to their corresponding bitlines to close SGD and boost the potential of corresponding string channel. The voltage bias for programming page 118 is shown in Figure 2(a) as an example. The programming process is typically realized by the incremental step pulse programming (ISPP) algorithm.[11] ISPP first injects electrons into floating gates to boost the V_{th} of programmed cells through FN mechanisms and then performs a verification to check whether the V_{th} has reached the desired level. If V_{th} is still lower than the desired voltage, the program-and-verify iteration will continue until the cell's V_{th} has reached the target level. Note that the NAND flash program operation can only add electrons into the floating gate and cannot remove them from the gate. As a result, the threshold voltage can only shift toward the right in Figure 1 during programming. The program operation is executed at page granularity.

Read

The read operation is also at the page granularity and the voltage bias is shown in Figure 2(b). The SGD, SGS, and all deselected wordlines are turned on. The wordline of selected read page is biased to a series of predefined reference voltages and the cell's threshold voltage can be determined to be between the most recent two read reference voltages when the cell conducts current.

Flash Memory Error Classification

We test the NAND flash memory using the cycle-by-cycle programming model shown in Figure 3. During each P/E cycle, the selected flash block is first erased. Then data are programmed into the block on a page granularity. Once a page has been programmed, it cannot be reprogrammed again unless the whole block is erased for the next P/E cycle. The stored data will be alive in the block until it becomes invalid. Before the stored data becomes invalid, it can be accessed multiple times. Once a page is programmed, we can test how long it retains data by reading the data value of the page after a *retention interval,* and comparing it to the original programmed value. Whether or not the data is retained correctly between two accesses depends on the time distance of two consecutive accesses. We repeat the above per-P/E-cycle procedure for thousands of cycles until the flash memory block becomes unreliable and reaches the end of its lifetime. Errors could happen in any stage of this testing process. We classify the observed errors into four different types, from the flash controller's point of view:

"We classify the observed errors into four different types, from the flash controller's point of view:."

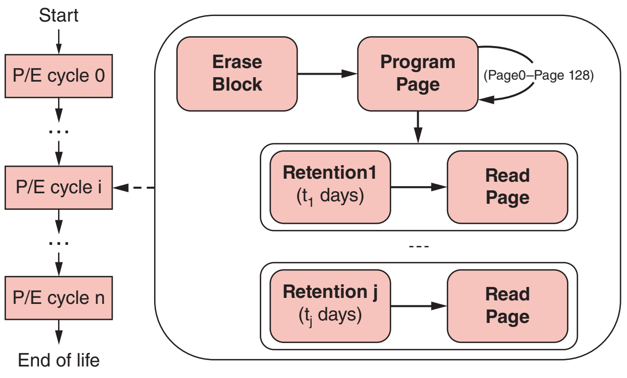


Figure 3: NAND flash programming model for error characterization
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

- *An erase error* happens when an erase operation fails to reset the cells to the erased state. This is mainly due to manufacturing process variations or defects caused by trapped electrons in the tunnel oxide after stress due to repeated P/E cycles.
- *A program interference error* happens when the data stored in a page changes (unintentionally) while a neighboring page is being programmed due to parasitic capacitance-coupling.
- *A retention error* happens when the data stored in a cell changes over time. The main reason is that the charge programmed in the floating gate may dissipate gradually through the leakage current.
- *A read error* happens when the data stored in a cell changes as a neighboring cell on the same string is read over and over.

"A retention error *happens when the data stored in a cell changes over time."*

Error Characterization Methodology

The following section describes the error characterization methodology.

Experimental Hardware

To characterize the error patterns, we built a hardware test platform that allows us to issue commands to raw flash chips without ECC.[2] The test platform mainly consists of three components: a HAPS–52 board with Xilinx Virtex-5 FPGAs used as NAND flash controller, a USB daughter board used to connect to the host machine, and a custom flash daughter board. The flash memory under test is a 2-bit MLC NAND flash device manufactured in 3x-nm technology. The device is specified to survive 3000 P/E cycles stress under 10-year data retention time if ECC with 4-bit error correction per 512 bits is applied. Details of the experimental flash test platform we use to collect our data are provided in [2].

"To characterize the error patterns, we built a hardware test platform that allows us to issue commands to raw flash chips without ECC."

Flash Error Testing Procedure

To test the P/E-cycle-dependence of errors, we stress-cycle flash memory blocks up to a certain number of erase cycles and check if the data is retained. This is achieved by iteratively erasing a block and programming pseudorandom data into it at room temperature.

We test whether the data is retained after T amount of time, to characterize retention errors. T is called the *retention test time* and is varied in the range of 1 day, 3 days, 3 weeks, 3 months, 1 year, and 3 years. We consider $T = \{1 \text{ day}, 3 \text{ days}\}$ to be short-term retention tests, while the remaining values of T are long-term retention tests. Short-term retention errors are characterized under room temperature. Long-term retention errors are characterized by baking the flash memory in the oven under 125° C. According to the classic temperature-activated Arrhenius law[12], the baking time at 125° C corresponds to about 450 times of the lifetime at room temperature (25° C).

We refer the reader to Cai et al.[3] for our testing and characterization methodology for program interference and read errors.

Error Characterization Results

We provide our experimental measurements of the errors in the state-of-the-art 3x-nm MLC NAND flash memory we have tested using our infrastructure. NAND flash errors show strong correlation with the number of P/E cycles, location of the physical cells, and the data values programmed into the cells. The following subsections analyze detailed error properties and briefly describe the causes of the observed phenomena. Our main focus in this article is retention errors, but our previous work analyzes all types of errors in detail[3], and we refer the reader to [3] for characterization and analysis of program interference, read, and erase errors.

Error Rate Analysis for Different Error Types

Figure 4 shows the bit error rate due to various types of NAND flash errors. The x-axis shows the number of P/E cycles and the y-axis depicts the raw bit error rate. Error rates are obtained characterized from the beginning of the flash chip's life until the region of >100x times of its specified lifetime (3000 P/E cycles for the chips we tested). We make several observations about error properties.

"...all types of errors are highly correlated with P/E cycles."

First, all types of errors are highly correlated with P/E cycles. At the beginning of the flash lifetime, the error rate is relatively low and the raw bit error rate is below 10^{-4}, within the specified lifetime (3K cycles). As the P/E cycles increase, the error rate increases exponentially. The P/E cycle-dependence of errors can be explained by the deterioration of the tunnel oxide under cycling stress. During erase and program operations, the electric field strength across the tunnel oxide is very high (for example, several million volts per centimeter). Such high electric field strength can lead to structural defects that trap electrons in the oxide layer. Over time, more and more defects accumulate and the insulation strength of the tunnel oxide degrades. As a result, charge can leak through the tunnel oxide and the threshold voltage of the cells can change more easily. This leads to more errors for all types of flash operations.

"The long-term retention errors are the most dominant; their rate is highest."

Second, there is a significant error rate difference between various types of errors. The long-term retention errors are the most dominant; their rate is highest. The program interference error rate ranks the second and is usually between error rates of 1-day and 3-day retention errors. The read error rate is slightly less than 1-day retention error rate, while the erase error rate is only around 7 percent of the read error rate.

Third, retention error rates are highly dependent on retention test time. If the time before we test for retention errors is longer, the floating gate of flash memory is more likely to lose more electrons through leakage current. This eventually leads to V_{th} shift across V_{th} windows and causes errors (see [6] for more detail). From our experimental data, we can see that the retention error rate increases linearly with the retention test time. For example, the 3-year retention error rate is almost three orders of magnitude higher than one-day retention error rate.

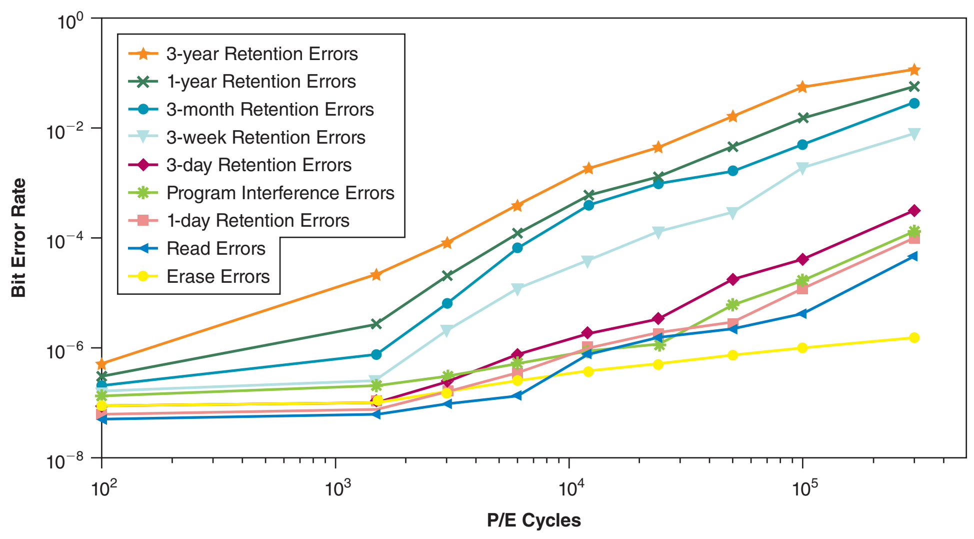


Figure 4: Rates of various types of errors as P/E cycles increase
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

Retention Error Analysis

Value dependence of retention errors: We find that the retention errors are value dependent; their frequency is asymmetric with respect to the value stored in the flash cell. Figure 5 demonstrates this asymmetric nature of retention errors by showing how often each possible value transition was observed due to an error. We characterized all possible error transitions, in the format *AB*→*CD*, where *AB* are the two bits stored in the cell before retention test, while *CD* are the two bits recorded in the cell after retention test. If the errors are not value dependent, the fraction of erroneous changes between each of the different value pairs should be equal. But, we find that this is not the case. The most common retention errors are 00→01, 01→10, 01→11 and 10→ 11, with their relative percentage over all retention errors being 46 percent, 44 percent, 5 percent, and 2 percent, respectively. The relative percentages among various error transitions are almost constant for different P/E cycles.

"We find that the retention errors are value dependent; their frequency is asymmetric with respect to the value stored in the flash cell."

To understand the reasons for value dependence, we need to observe Figure 1 in conjunction with the value transition observed in the most common retention errors. We find that the most common retention errors (00→01, 01→10, 01→11, and 10→ 11) are all cases in which V_{th} shifts towards the left (see Figure 1). This can be explained by an understanding of the retention error mechanisms. During retention test, the electrons stored on the floating gate gradually leak away under stress induced leakage current (SILC). When the floating gate loses electrons, its V_{th} shifts left from the state with more electrons to the state with fewer programmed electrons (as seen in Figure 1, states to the left have fewer electrons trapped on the gate than states to the right). It is significantly less likely for the cells to shift right in the opposite direction because this requires the addition of more electrons. As the states of 00 and 01 hold the largest number of electrons on the floating gates, SILC is higher in these states and therefore it is more likely for the V_{th} of the cells in these two states to shift left, which leads to the observation that most common errors are due to shifting from these states (00→01, 01→10, 01→11).

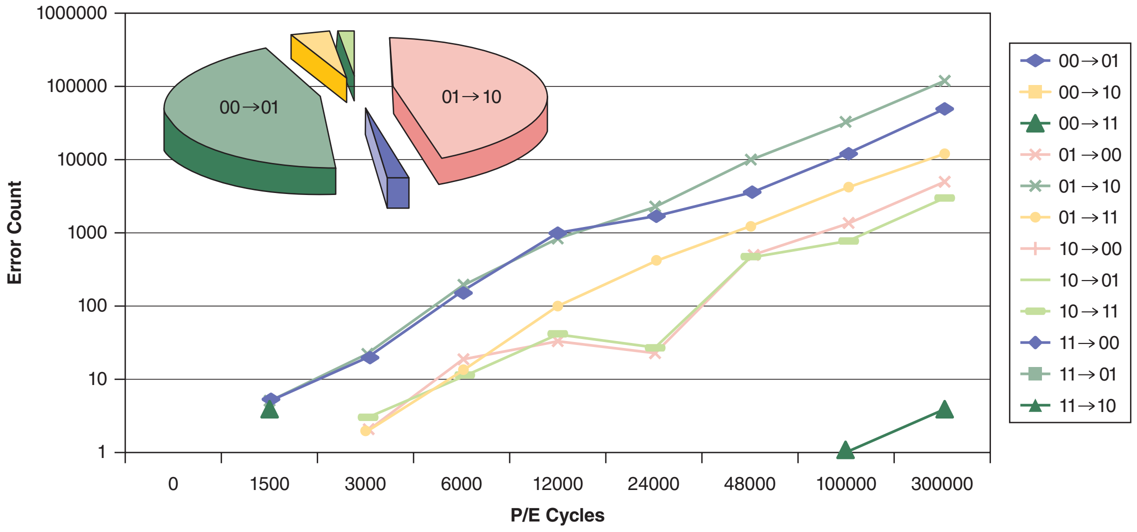


Figure 5: Value dependence of retention errors
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

"We also characterized the relation between retention errors and their physical locations."

Location dependence of retention errors: We also characterized the relation between retention errors and their physical locations. The experimental results are shown in Figure 6. The x-axis shows the wordline number of a block and the y-axis shows the bit error rates of pages on the corresponding wordline (observed after 50K P/E cycles). Each wordline contains four pages, including LSB-even, LSB-odd, MSB-even, and MSB-odd. The bit error rates of these four types of pages are shown in Figure 6. Several major observations are in order.

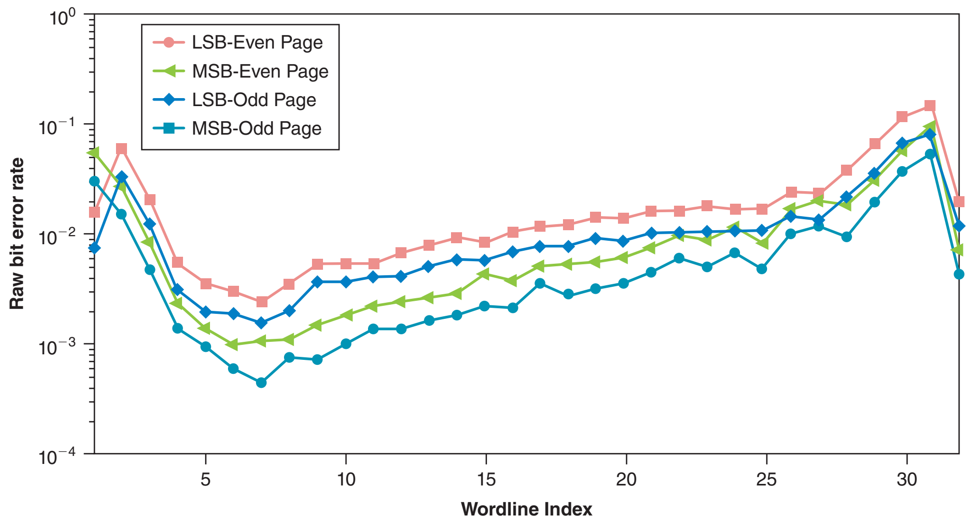


Figure 6: Retention error rate vs. physical location
(Source: Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, 2012[3])

First, the error rate of the MSB page is higher than that of the corresponding LSB page. In our experimental data, the MSB-even page error rate is 1.88 times higher than the LSB-even page error rate and the MSB-odd page error rate is 1.67 times higher than the LSB-odd page error rate on average. This phenomenon can be explained by understanding the bit mapping within the flash memory. Dominant retention errors are mainly due to the shifting of V_{th} between two adjacent threshold voltage levels, that is shifting of V_{th} from the ith level to the (i–1)th level. From the bit mapping in Figure 1, we can see that such a V_{th} shift can cause an LSB error only at the border REF2 between state L2 (01) and state L1 (10) because these are the only two adjacent threshold voltage levels where LSB differs. On the other hand, such a V_{th} shift can cause an MSB error on any border (REF1, REF2, REF3) between any two adjacent states because MSB differs between all possible adjacent threshold voltage levels. Hence, since the likelihood of a change in MSB when a V_{th} shift happens between adjacent states is higher than the likelihood of a change in LSB, it is more common to see retention errors in MSB than in LSB.

"...the error rate of the MSB page is higher than that of the corresponding LSB page."

Second, the retention error rate of odd pages is always higher than that of the corresponding even pages. For example, the error rate of MSB-odd pages is 2.4 times higher than that of MSB-even pages, and the error rate of LSB-odd pages is 1.61 times higher than that of LSB-even pages, on average. This result can be explained by the over-programming introduced by inter-page interference. Generally, the pages inside a flash block are programmed sequentially, and a block is programmed in order, that is, from page 0 to page 127. For the same wordline, even pages are programmed first followed by odd pages. When odd pages are programmed, a high positive program voltage is applied to the control gates of all the cells on the wordline, including the cells of the even page, which has already been programmed. Thus, the even page comes under programming current disturbance and some additional electrons could be attracted into the floating gates of the even page. As a result of this, the V_{th} of cells of the even pages shift slightly to the right. Consequently, the cells of the even pages hold more electrons than the cells of the odd pages, even if they are programmed to the same logic value and are in the same threshold voltage window (in some sense, the cells of the even pages are thus more resistant to leakage because they hold more electrons). When electrons leak away over time during the retention test, as a result, it is more likely for the cells of even pages to still keep their original threshold voltage window and hold the correct value. In contrast, since the cells of the odd pages hold fewer electrons, they are more likely to transition to a different threshold voltage window and hence acquire an incorrect value as electrons leak over time.

"...the retention error rate of odd pages is always higher than that of the corresponding even pages."

Third, the bit error rates of all the four types of pages have the same trend related to physical wordlines. For example, the error rates of the four types of pages are all high on wordline #31 and are all low on wordline #7. We conclude that error rates are correlated with wordline locations. This could possibly be due to process variation effects, which could be similar across the same wordline.

"The major takeaway from our measurement and characterization results is that the rate of retention errors, which are the most common form of flash errors, is asymmetric in both original cell value and the location of the cell in flash bit organization."

"The basic idea of the FCR schemes is to periodically read, correct, and refresh (reprogram or remap) the stored data before it accumulates more retention errors than can be handled by ECC."

The major takeaway from our measurement and characterization results is that the rate of retention errors, which are the most common form of flash errors, is asymmetric in both original cell value and the location of the cell in flash bit organization. This observation can potentially be used to devise error protection or correction mechanisms that have varying strength based on cell value and location.

Mitigating Retention Errors: Flash Correct-and-Refresh

We describe a set of new techniques, called Flash Correct-and-Refresh (FCR), that exploit the dominance and characteristics of retention errors to significantly increase NAND flash lifetime while incurring minimal overhead. The basic idea of the FCR schemes is to periodically read, correct, and refresh (reprogram or remap) the stored data before it accumulates more retention errors than can be handled by ECC. Thus, we can achieve a low uncorrectable bit error rate (UBER) while still using a simple, low-overhead ECC. Two key questions central to designing a system that uses FCR techniques are: (1) *how* to refresh the data in flash memory and (2) *when* to refresh the data. We address the first question with two techniques for how to refresh the data: remapping (in the section "Remapping-based FCR Mechanisms") and reprogramming in-place (in the section "In-Place Reprogramming-based FCR Mechanisms"). We then tackle the second question with two techniques for when to refresh: periodically and adaptively based on the number of P/E cycles (Section 6.3).

Remapping-based FCR Mechanisms

Unlike DRAM cells, which can be refreshed in-place[13], flash cells generally must first be erased before they can be programmed. To remove the slow erase operation from the critical path of write operations, current wear-leveling algorithms remap the data to another physical location rather than erasing the data and then programming in-place. The flash controller maintains a list of free blocks that have been erased in background through garbage collection and are ready for programming. Whenever a write operation is requested, the controller's wear-leveling algorithm selects a free block and programs it directly, remapping the logical block address to the new physical block.

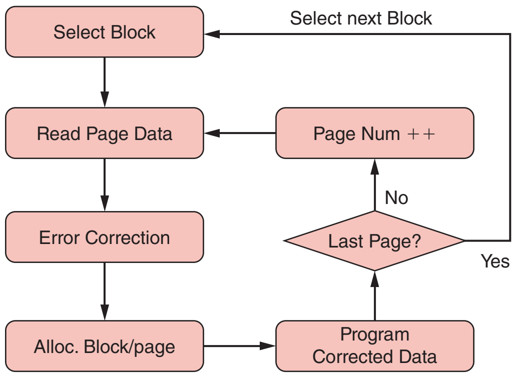


Figure 7: Operation of a remapping-based flash correct-and-refresh scheme (Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

The key idea of remapping-based FCR is to leverage the existing wear-leveling mechanisms to periodically read, correct, and remap to a different physical location each valid flash block in order to prevent it from accumulating too many retention errors. Figure 7 shows the operational flow of remapping-based FCR: (1) During each refresh interval, a block with valid data that needs to be refreshed is selected. (2) The valid data in the selected block is read out page by page and moved to the SSD controller. (3) The ECC engine in the SSD controller corrects all the errors in the read data, including retention errors that have accumulated since the last refresh. After ECC, the data are error free. (4) A new free block is selected and the error free data are programmed to the new location, and the logical address is remapped. Note that the proposed address remapping techniques leverage existing hardware and software of contemporary wear-leveling and garbage collection algorithms.

Unfortunately, periodic remapping of every block introduces additional erase cycles. This is because after the flash data are corrected and remapped to the new location, the original block is marked as outdated. Thus, the block will eventually be erased and reclaimed by garbage collection. The more frequent the remap operations, the more the additional erase operations, which wears out flash memory faster. As such, there might be an inflection point beyond which increasing the refresh rate in remapping-based FCR can lead to reduced lifetime. To avoid this potential problem, we next introduce enhanced FCR methods, which minimize unnecessary remap operations.

In-Place Reprogramming-based FCR Mechanisms

To reduce the overhead associated with periodic remapping, we describe a technique for periodic *in-place reprogramming* of the block most of the time, without a preceding erase operation, which can greatly reduce the overhead of periodic remapping. This in-place reprogramming takes advantage of the key observation that retention errors arise from the loss of electrons on the floating gate over time and *the flash cell with retention errors can be reprogrammed to its original correct value without an erase operation* using the incremental step pulse programming (ISPP) scheme used to program flash memory. We first provide background on ISPP.

"...the flash cell with retention errors can be reprogrammed to its original correct value without an erase operation *using the incremental step pulse programming (ISPP) scheme used to program flash memory."*

ISPP

Before a flash cell can be programmed, the cell must be erased (that is, all charge is removed from the floating gate, setting the threshold voltage to the lowest value). When a NAND flash memory cell is programmed, a high positive voltage applied to the control gate causes electrons to be injected into the floating gate. The threshold voltage of a NAND flash cell is programmed by injecting a precise amount of charge onto the floating gate through ISPP.[11] During ISPP, floating gates are programmed iteratively using a step-by-step program-and-verify approach. After each programming step, the flash cell threshold voltage is boosted up. Then, the threshold voltage of the programmed cells are sensed and compared to the target values. If the cell's threshold voltage level is higher than the target value, the program-and-verify iteration will stop. Otherwise the flash cells are programmed once again and more electrons are added to the floating gates to boost the threshold voltage. This program-and-verify cycle continues iteratively until all the cells' threshold voltages reach the target values. Using ISPP, flash memory cells can only be programmed from a state with fewer electrons to a state with more electrons and cannot be programmed in the opposite direction.

Retention Error Mechanisms

Retention errors are caused by the loss of electrons from the floating gate over time. As such, a cell with retention errors moves from a state with more electrons to a state with fewer electrons. Figure 8(a) shows the relative relationship between the stored data value and its corresponding threshold voltage distribution for a typical MLC flash storing 2-bits per cell. The leftmost state is the erased state (state 11) with the smallest threshold voltage, and there is no charge on the floating gate. The states located on the right in Figure 8(a) are programmed with more electrons and have higher threshold voltages than the states located relatively to the left. Over time, as the electrons on the floating gate leak away, the threshold voltage of a cell shifts to the left, as shown in Figure 8(b). If the threshold voltage of a cell shifts too far to the left (that is, it loses too many electrons from the floating gate), it will cross the read reference voltage between adjacent states and can be misinterpreted during a read as the wrong value.

"Over time, as the electrons on the floating gate leak away, the threshold voltage of a cell shifts to the left..."

In-Place Reprogramming Can Fix Retention Errors

A cell with a retention error can be reprogrammed to the value it had before the floating gate lost charge by recharging additional electrons onto the floating gate through ISPP, as shown in Figure 8(c). Note that this does not require an erase operation because the only objective is to add more electrons (not to remove them), which can be accomplished by simple programming.

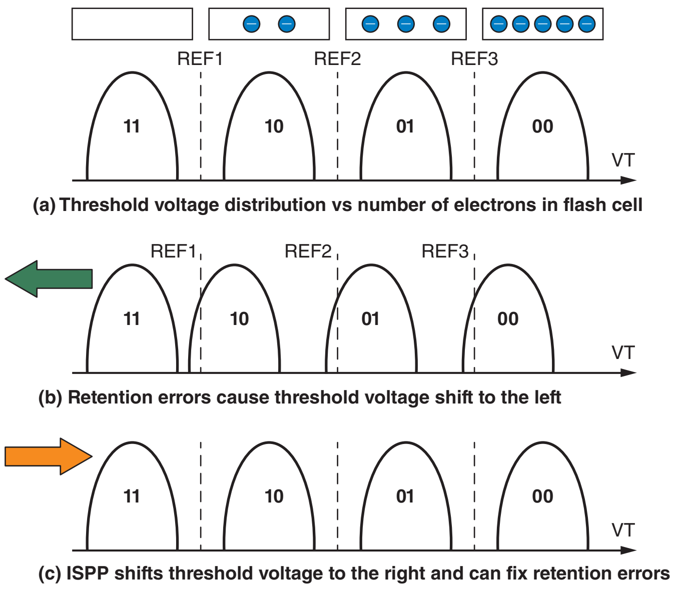

REF1
REF2
REF3
11
10
01
00
VT
(a) Threshold voltage distribution vs number of electrons in flash cell
(b) Retention errors cause threshold voltage shift to the left
(c) ISPP shifts threshold voltage to the right and can fix retention errors

Figure 8: Retention errors are caused by threshold voltage shift to the left and can be fixed by programming in-place using ISPP
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

6.2.4 Basic In-Place Reprogramming-based FCR Mechanism

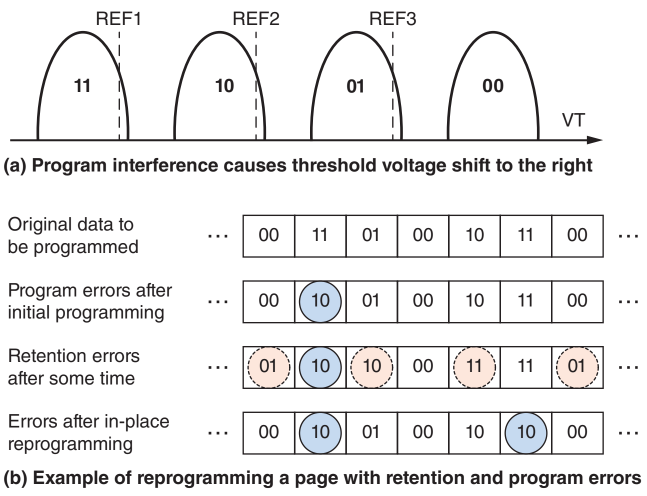
A basic FCR mechanism that uses in-place reprogramming works as follows. Periodically, a block is selected to be refreshed and read page-by-page into the flash controller. By selecting a suitable refresh interval, we can ensure that the total error number is below the correction capability of the ECC. Then we can reprogram the flash cells in the same location with the error-corrected data, without erasing the whole block. If the new corrected value corresponds to a state with more charge than the old value, then the cell can be in-place reprogrammed to the correct value. If the corrected value is exactly the same as the original value, in-place reprogramming will not change the stored data value, as ISPP will stop programming the cell as soon as it detects that the target value has already been reached. Note that most of the cells are reprogrammed with exactly the same data value as error rates are generally significantly below 1 percent.

Problem: Accumulated Program Errors

While this basic mechanism can effectively fix retention errors, it introduces a problem because there is another error mechanism in flash cells that is caused by program operations, which are required to perform in-place reprogramming. When a flash cell is being programmed, additional electrons may be injected into the floating gates of its neighbor cells due to coupling capacitance.[14] The threshold voltage distribution of the neighbor cells will shift *right* as they gain more electrons, as shown in Figure 9(a). If the threshold voltage shifts right by too much, it will be misread as an error value that represents a state located to the right. This is called a program interference error (or simply a program error). Although it is a less common error mechanism than retention errors as we have shown in Figure 4, periodic reprogramming can exacerbate the effects of program errors.

"When a flash cell is being programmed, additional electrons may be injected into the floating gates of its neighbor cells due to coupling capacitance."

Two potential issues are: (1) As ISPP cannot remove electrons from the floating gate, program errors cannot be fixed by in-place reprogramming; (2) Reprogramming of a page can introduce additional program errors due to the additional program operations. Figure 9(b) illustrates both issues in the context of in-place programming. First, the original data is programmed into the page. This initial programming can cause some program errors (for example, value 11 is programmed as 10 on the second cell from the left). After some time, retention errors start to appear in the stored data (for example, the first cell changes from state 00 to 01). Note that there are generally many more retention errors than program errors. When the page is reprogrammed in-place, it is first read out and corrected using ECC. The error-corrected data (which is the same as the original data) is then written back (programmed) into the page. This corrects all the retention errors by recharging the cells that lost charge. However, this reprogramming does not correct the program error (in the second cell) because this correction requires the removal of charge from the second cell's floating gate, which is not possible without an erase operation. Furthermore, additional program errors can appear (for example, in the sixth cell) because the in-place program operation can cause additional disturbance.


REF1
REF2
REF3
11
10
01
00
VT
Original data to be programmed
… 00 11 01 00 10 11 00 …
Program errors after initial programming
… 00 10 01 00 10 11 00 …
Retention errors after some time
… 01 10 10 00 11 11 01 …
Errors after in-place reprogramming
… 00 10 01 00 10 10 00 …

(a) Program interference causes threshold voltage shift to the right

(b) Example of reprogramming a page with retention and program errors

Figure 9: In-place reprograming can correct retention errors but not program errors because in-place programming can only add more electrons into the floating gate and cannot remove them. Note that red values with dotted circles are retention errors and blue ones with solid circles are program errors
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

"To mitigate the errors accumulated due to periodic reprogramming, we propose a hybrid reprogramming/ remapping-based FCR technique to control the number of reprogram errors."

Hybrid FCR

To mitigate the errors accumulated due to periodic reprogramming, we propose a hybrid reprogramming/remapping-based FCR technique to control the number of reprogram errors. The key idea is to monitor the right-shift error count present in each block. If this count is below a certain threshold (likely most of the time) then in-place reprogramming is used to correct retention errors. If the count exceeds the threshold, indicating that the block has too many accumulated program errors, then the block is remapped to another location, which corrects both retention and program errors. In our evaluation, we set the threshold to 30 percent of the maximum number of errors that could be corrected by ECC, which is conservative. Figure 10 provides a flowchart of this hybrid FCR mechanism. Note that this hybrid FCR mechanism greatly reduces the additional erase operations present in remapping based FCR because it remaps a block (requires an erase operation) only when the number of accumulated program errors is high, which is rare due to the low program error rate.

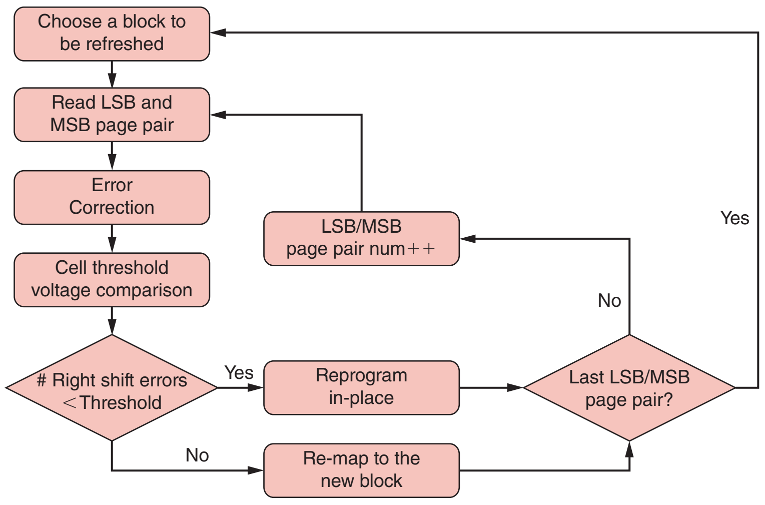

Choose a block to be refreshed
Read LSB and MSB page pair
Error Correction
Cell threshold voltage comparison
# Right shift errors < Threshold
Yes
No
Reprogram in-place
Re-map to the new block
Last LSB/MSB page pair?
No
Yes
LSB/MSB page pair num++

Figure 10: Hybrid FCR workflow: if reprogramming error count is less than a threshold, in-place reprogram the block; otherwise, remap to a new block
(Source: Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, 2012[4])

Adaptive-Rate FCR

So far we assumed that FCR mechanisms, be it based on in-place reprogramming or remapping, are invoked periodically. This need not be the case. In fact, we observe that the rate of (retention) errors is very low during the beginning of flash lifetime, as shown in Figure 4. Until more than 1000 P/E cycles, the retention error rate is lower than the acceptable raw BER that can be corrected by the simplest BCH code (not shown, but described in detail in [4]), which is a commonly used ECC type in flash memories. Hence, at the beginning of its lifetime, flash memory does not need to be refreshed. Retention error rate increases as the number of P/E cycles increases. We leverage this key observation to reduce the number of unnecessary refresh operations.

The main idea of adaptive-rate FCR is to adapt the refresh rate to the number of P/E cycles a block has incurred. Initially, refresh rate for a block starts out at zero (no refresh). Once ECC becomes incapable of correcting retention errors, the block's refresh rate increases to tolerate the increased retention error rate. Hence, refresh rate is gradually increased over each flash block's lifetime to adapt to the increased P/E cycles. The whole lifetime of a flash block can be divided into intervals with different refresh rates ranging, for example, from no refresh (initially), yearly refresh, monthly refresh, weekly refresh, to daily refresh. The frequency of refresh operations at a given P/E cycle count is determined by the acceptable raw BER provided by the used ECC and the BER that corresponds to the P/E cycle count (which can be known by the controller[4]). Note that this mechanism requires keeping track of P/E cycles incurred for each block, but this information is already maintained to implement current wear-leveling algorithms.

"…refresh rate is gradually increased over each flash block's lifetime to adapt to the increased P/E cycles."

Additional Considerations

We briefly discuss some additional factors that affect the implementation and operation of the proposed FCR mechanisms.

Implementation Cost

The FCR mechanisms do not require hardware changes. They require changes in FTL software/firmware to implement the flowcharts shown in Figures 7 and 10. FCR can leverage the per-block validity and P/E cycle information that is already maintained in existing flash systems to implement wear leveling.

"The FCR mechanisms do not require hardware changes."

Power Supply Continuity

To perform a refresh, the flash memory must be powered. As FCR is proposed mainly 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.

Response Time Impact

Refresh may interfere with normal flash operations and degrade the response time. To reduce this penalty, we can decrease the refresh priority, making it run in the background. The SSD can issue refresh operations whenever it is idle, and refresh operations can be interrupted to avoid the impact on the response time of normal operations. Unlike DRAM, where refresh is triggered frequently (for example, every 64 ms) to maintain correctness[13], the refresh period of FCR is at least a day, and the SSD can finish refresh operations within the refresh period. Recent work has shown that the response time overhead is within a few percent for daily refresh.[15] Note that our hybrid and adaptive FCR techniques have much lower overhead for refresh operations than periodic remapping based FCR.

"Unlike DRAM, where refresh is triggered frequently (for example, every 64 ms) to maintain correctness, the refresh period of FCR is at least a day..."

Additional Erase Cycles

FCR introduces additional erase operations. Our evaluations take into account the impact of additional erase operations on flash lifetime and energy consumption.

Adapting to Variations in Retention Error Rate

Note that retention error rate is usually constant for a given refresh rate and P/E cycle combination. However, there are environmental factors, such as temperature, that can change this rate. For example, retention error rate would be dependent on temperature. To adapt to dynamic fluctuations in retention error rate, our hybrid FCR and adaptive-rate FCR mechanisms monitor the changes in the retention error rate at periodic intervals, and increase or decrease the refresh (that is, FCR) rate if the error rate in the previous interval is greater or less than a threshold. These mechanisms are similar in principle to what is employed in DRAM to adapt refresh rate to temperature changes.[13]

Evaluation of Flash Correct-and-Refresh

We evaluate FCR using Disksim[20] with SSD extensions[21]. All proposed techniques are simulated using various I/O traces from real workloads: *iozone*[22], *cello99*[23], *oltp*, *postmark*[24], *MSR-Cambridge*[25] and a *web search engine*[26]. We configure the simulated flash-based SSD with four channels. Each channel has eight flash chips. Each flash chip has 8,192 blocks containing 128 pages. The page size is 8 KB. The total storage capacity is 256 GB. The energy of flash read, program, and erase operations are collected from our experimental flash memory platform[2], and are used in the simulation infrastructure to obtain the overall energy consumption. The details of our experimental evaluation methodology, workloads, and our method for estimating lifetime are described in our previous work[4]. We present the major results showing the effect of our mechanisms on flash lifetime and energy consumption in this article. Much more detailed analyses of our individual techniques, analysis of sensitivity to refresh interval length, and results on individual workloads are provided in [4].

Effect on Flash Memory Lifetime

Figure 11 shows the lifetime improvement provided by three different versions of FCR compared to the baseline with no refresh. The adaptive-rate FCR mechanism is implemented on top of the hybrid FCR substrate. Flash lifetime is evaluated under various ECC configurations, ranging from weak 512-bit to strong 32-kb BCH codes (described and evaluated in detail in [4]). The refresh period of each periodic mechanism is chosen on a per-workload basis such that the lifetime provided for a workload by the mechanism is maximized (more analysis on the refresh period can be found in [4]). Adaptive-rate FCR, which adaptively and realistically chooses the refresh period, provides the highest lifetime improvement over the baseline as it corrects retention errors while avoiding unnecessary refreshes. The improvements are especially significant in read-intensive workloads since these workloads do not have high P/E cycles, causing the adaptive-rate FCR to keep the refresh rate very low. On average, adaptive-rate FCR provides 46.7x, 4.8x, and 1.5x higher flash lifetime compared to no-refresh (on the baseline system using 512-bit ECC), remapping-based FCR, and hybrid FCR, respectively. Note that the lifetime improvement provided by the much stronger 32-kb ECC is only four times that of the lifetime provided by the baseline 512-bit ECC, yet the implementation of the former, stronger ECC, requires 71 times the power consumption and 85 times the area of the latter, weaker ECC.[4] Contrast this with the 46.7x lifetime improvement provided by adaptive-rate FCR on the system with 512-bit ECC. Thus, improving lifetime via FCR is much more effective and efficient than doing so by increasing the strength of ECC. We conclude, based on these results, that adaptive-rate FCR implemented over the hybrid FCR mechanism is a promising mechanism for significant lifetime enhancement of flash memory at low cost.

"...adaptive-rate FCR provides 46.7x, 4.8x, and 1.5x higher flash lifetime compared to no-refresh"...

"...improving lifetime via FCR is much more effective and efficient than doing so by increasing the strength of ECC."

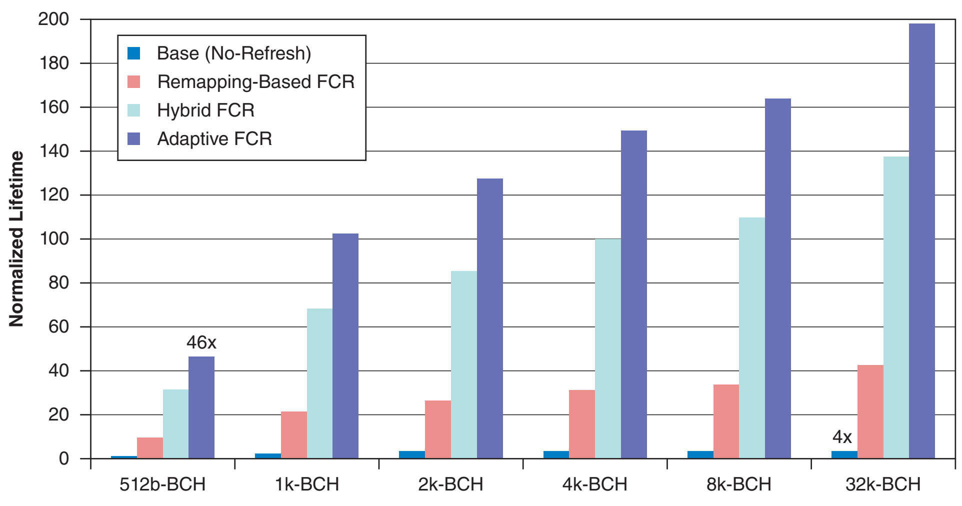


Figure 11: Lifetime improvement provided by different FCR techniques in comparison to systems employing varying strength ECC (BCH) codes. Data normalized to lifetime with no refresh on a system with 512-bit ECC
(Source: Onur Mutlu, 2012)

P/E Cycle and Energy Overhead Analysis

FCR techniques can introduce two main overheads: (1) additional P/E cycles due to remapping; (2) additional energy consumed by refresh operations. A detailed evaluation of the former is provided in [4]. Note that all P/E cycle overheads have already been accounted for in the collection of the flash lifetime results.

Figure 12 shows the additional flash energy consumption of remapping-based FCR and hybrid FCR averaged over all workloads compared to a system with no FCR. The refresh energy is estimated under the worst-case scenario that all data are to be refreshed. Even if we assume we must refresh the *entire* SSD each day, the energy overhead is only 7.8 percent and 5.5 percent for remapping-based FCR and hybrid FCR respectively. When the refresh interval is three weeks, the energy overhead is almost negligible (less than 0.4 percent). We also observe that hybrid FCR has less energy overhead than remapping based FCR mainly because hybrid FCR reduces the high-energy erase/remap operations by performing in-place reprogramming most of the time.

"When the refresh interval is three weeks, the energy overhead is almost negligible..."

We also evaluate the energy overhead of adaptive-rate FCR and find that it is only 1.5 percent (not shown in the figure). Recall that adaptive-rate FCR starts out with no refresh and gradually increases the refresh rate up to daily refresh as the P/E cycles accumulate. Yet its energy overhead is significantly lower than periodic daily refresh. We conclude that adaptive-rate FCR is the most superior of flash correct-and-refresh mechanisms in terms of both lifetime and energy consumption.

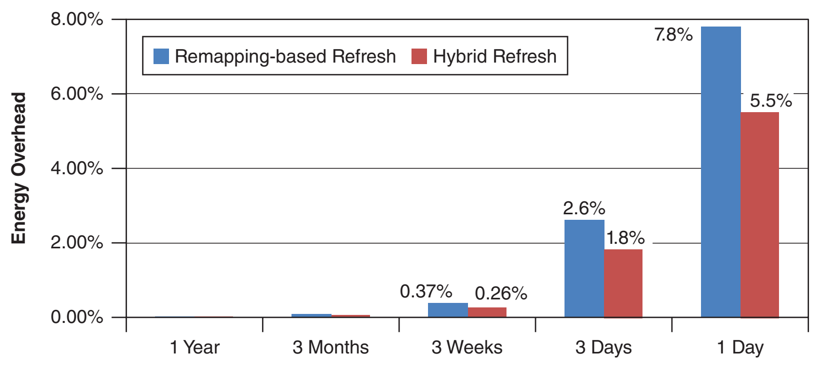


Figure 12: Energy increase of remapping-based and hybrid FCR vs. no refresh
(Source: Onur Mutlu, 2012)

Ongoing Work

In our comprehensive continued effort for enhancing flash memory scaling to smaller technology nodes, we have been characterizing the effects of different error mechanisms in flash memory, developing models to predict how they change over the lifetime of flash memory, and designing error tolerance mechanisms based on the developed characterization and models.

Recently, we have also experimentally investigated and characterized the threshold voltage distribution of different logical states in MLC NAND flash memory.[6] We have developed new models that can predict the shifts in the threshold voltage distribution based on the number of P/E cycles endured by flash memory cells. Our key results, presented in [6], show that 1) the threshold voltage distribution of flash cells that store the same value can be approximated, with reasonable accuracy, as a Gaussian distribution, 2) under ideal wear leveling, the flash cell can be modeled as an AWGN (Additive White Gaussian Noise) channel that takes the input (programmed) threshold voltage signal and outputs a threshold voltage signal with added Gaussian white noise, and 3) threshold voltage distribution of flash cells that store the same value gets distorted (shifts to right and widens around the mean value) as the number of P/E cycles increases. This distortion can be accurately modeled and predicted as an exponential function of the P/E cycles, with more than 95 percent accuracy. Such predictive models can aid the design of much more sophisticated error correction methods, such as LDPC codes[7], which are likely needed for reliable operation of future flash memories. We refer the reader to [6] for more detail.

"We have developed new models that can predict the shifts in the threshold voltage distribution based on the number of P/E cycles endured by flash memory cells."

We are currently investigating another increasingly more significant obstacle to continued MLC NAND flash scaling, which is the increasing cell-to-cell program interference due to increasing parasitic capacitances between the cells' floating gates. Accurate characterization and modeling of this phenomenon are needed to find effective techniques to combat this program interference. In our recent work[16], we leverage the *read retry* mechanism found in some flash designs to obtain measured threshold voltage distributions results from state-of-the-art 2Y-nm (24- to 20-nm) MLC NAND flash chips. These results are then used to characterize the cell-to-cell program interference under various programming conditions. We show that program interference can be accurately modeled as additive noise following Gaussian-mixture distributions, which can be predicted with 96.8 percent accuracy using linear regression models. We use these models to develop and evaluate a read reference voltage prediction technique that reduces the raw flash bit error rate by 64 percent and increases the flash lifespan by 30 percent. We refer the reader to [16] for more detail.

Finally, apart from investigating scaling challenges in flash memory, we are investigating techniques to enable better scaling of DRAM. Improving DRAM cell density by reducing the cell size, as has been done traditionally, is becoming significantly more difficult due to increased manufacturing complexity/cost and reduced cell reliability. We are examining alternative ways of enhancing the performance and energy-efficiency of DRAM while still maintaining low cost. A key direction is to co-design the DRAM controller and DRAM, rethinking the DRAM interface and microarchitecture, such that DRAM scaling challenges are tolerated at the system level. For example, we have recently proposed new techniques to reduce DRAM access latency at low cost by segmenting bitlines and creating a low-latency low-energy segment within a subbank[17], to increase DRAM parallelism and locality by enabling pipelined access of subbanks and enabling multiple row buffers to be concurrently active within a bank[18], to reduce the number of DRAM refreshes by taking advantage of variation in retention times of DRAM rows in a low-cost manner[13], and to accelerate bulk data copy and initialization operations by performing them solely in DRAM with only minor modifications to DRAM[19]. We have also experimentally characterized retention behavior of DRAM cells and rows for 248 commodity DRAM chips[27], with the goal of developing mechanisms that can dynamically profile retention times of different rows. We observed two significant phenomena: data pattern dependence, where the retention time of DRAM cells is significantly affected by the data stored in other DRAM cells, and variable retention time, where the retention time of some DRAM cells changes over time. We refer the reader to these respective works for further detail.

"A key direction is to co-design the DRAM controller and DRAM, rethinking the DRAM interface and microarchitecture, such that DRAM scaling challenges are tolerated at the system level."

Conclusion

Reliability and energy efficiency challenges posed by technology scaling are a critical problem that jeopardizes both flash memory and DRAM capacity, cost, performance, lifetime, and efficiency. In this article, we have described our recent error analysis of flash memory and a new method to improve flash memory lifetime. We hope other works by us and other researchers in the field collectively enable the memory and microprocessor industry to develop cooperative techniques to enable scalable, efficient, and reliable flash memory (and DRAM) that continues to scale to smaller feature sizes.

References

[1] A. Maislos et al., "A New Era in Embedded Flash Memory," FMS 2011.

[2] Y. Cai, et al., "FPGA-Based Solid-State Drive Prototyping Platform," FCCM 2011.

[3] Y. Cai et al., "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization and Analysis," DATE 2012.

[4] Y. Cai et al., "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," ICCD 2012.

[5] Y. Koh, "NAND Flash Scaling Beyond 20nm," IMW 2009.

[6] Y. Cai et al., "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling," DATE 2013.

[7] R. G. Gallager, *Low-Density Parity Check Codes*. Cambridge: MIT Press, 1963.

[8] T. Hara, et al., "A 146-mm2 8-Gb multi-level NAND flash memory with 70-nm CMOS technology," JSSC, Vol. 41, pp. 161–169, 2006.

[9] Y. Li, et al., "A 16Gb 3-Bit Per Cell(X3) NAND Flash Memory on 56nm Technology With 8MB/s Write Rate," JSSC, Vol. 44, pp. 195–207, 2009.

[10] N. Shibata, et al., "A 70nm 16Gb 16-Level-Cell NAND flash Memory," JSSC, Vol. 43, pp. 929–937, 2008.

[11] K. D. Suh, et al., "A 3.3V 32Mb NAND Flash Memory with Incremental Step Pulse Program Scheme," JSSC, Vol. 30, No.11, pp. 1149–1156, 1995.

[12] M. Xu, et al., "Extended Arrhenius law of time-to-breakdown of ultrathin gate oxides," Applied Physics Letters, Vol. 82, pp. 2482–2484, 2003.

[13] J. Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

[14] K. Park et al., "A Zeroing Cell-to-Cell Interference Architecture with Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories," JSSC 2008.

[15] Y. Pan et al., "Quasi-Nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System Performance for Enterprise Applications," HPCA 2012.

[16] Y. Cai et al. "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Application," Carnegie Mellon University SAFARI Technical Report, January 2013.

[17] D. Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

[18] Y. Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.

[19] V. Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," Carnegie Mellon University SAFARI Technical Report, March 2013.

[20] J. Bucy et al., "DiskSim Simulation Environment Reference Manual," 2008.

[21] N. Agrawal et al., "Design Tradeoffs for SSD Performance," USENIX 2008.

[22] IOzone.org, "IOzone Filesystem Benchmark," http://iozone.org.

[23] Open Source software at HP Labs, http://tesla.hpl.hp.com/opensource.

[24] J. Katcher, "Postmark: a New File System Benchmark Technical Report," 1997.

[25] SNIA: IOTTA Repository, http://iotta.snia.org/tracetypes/3.

[26] UMass Trace: http://traces.cs.umass.edu/index.php/Storage/Storage.

[27] J. Liu et al., "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," To Appear in ISCA 2013.

Author Biographies

Yu Cai obtained his PhD degrees in Electrical and Computer Engineering from Carnegie Mellon University (2012). He received an MS degree in Electronic Engineering from Tsinghua University and a BS degree in Telecommunication Engineering in Beijing University of Posts and Telecommunication (BUPT). Currently, he is a staff engineer, SSD Architect working in the Flash Channel Department of LSI Corporation. Prior to LSI, he worked at Hong Kong Applied Science and Research Institute (ASTRI), Lucent Technology, and Microsoft Research Asia (MSRA). His research interests include data storage, reconfigurable computing, and wireless communication.

Gulay Yalcin is a PhD student at Universitat Politecnica de Catalunya and a researcher student in Bacelona Supercomputing Center. She holds a BS degree in Computer Engineering from Hacettepe University and an MS degree in Computer Engineering from TOBB University of Economics and Technology. Her research interests are reliability and energy minimization in computer architecture. For more information please see the web page at http://www.bscmsrc.eu/people/gulay-yalcin.

Onur Mutlu is the Dr. William D. and Nancy W. Strecker Early Career Professor at Carnegie Mellon University. He enjoys teaching and researching important and relevant problems in computer architecture and computer systems, including problems related to the design of memory systems, multi-core architectures, and scalable and efficient systems. He obtained his PhD and MS in ECE from the University of Texas at Austin (2006) and BS degrees in Computer Engineering and Psychology from the University of Michigan, Ann Arbor. Prior to Carnegie Mellon, he worked at Microsoft Research (2006-2009), Intel Corporation, and Advanced Micro Devices. He was a recent recipient of the IEEE Computer Society Young Computer Architect Award, CMU College of Engineering George Tallman Ladd Research Award, Intel Early Career Faculty Honor Award, Microsoft Gold Star Award, best paper awards at ASPLOS, VTS and ICCD, and a number of "computer architecture top pick" paper selections by the IEEE Micro magazine. For more information, please see his web page at http://www.ece .cmu.edu/~omutlu.

Erich F. Haratsch is Director of Engineering, Flash Channel Technology at LSI Corporation. In this role, he leads the development of advanced signal processing and error correction coding features for solid-state disk controllers. Prior to joining LSI, Haratsch was a Senior Member of Technical Staff at Agere Systems, where he developed signal processing architectures for magnetic recording in hard disk drives. He also developed equalizer and decoder architectures for Gigabit Ethernet over copper and optical communications at Bell Labs Research. Haratsch is the author of more than 30 peer-reviewed journal and conference papers, and holds 35 U.S. patents. He is a Senior Member of IEEE. Haratsch earned his MS and PhD from the Technical University of Munich, Germany.

Adrián Cristal is co-manager of the Computer Architecture for Parallel Paradigms research group at BSC. His interests include high-performance microarchitecture, multi- and many-core chip multiprocessors, transactional memory, programming models, and computer architectures for Big Data. He received a PhD from the Computer Architecture Department at the Polytechnic University of Catalonia (UPC), Spain, and he has a BS and an MS in computer science from the University of Buenos Aires, Argentina.

Osman Unsal received the BS, MS, and PhD degrees in Electrical and Computer Engineering from Istanbul Technical University (Turkey), Brown University (USA) and University of Massachusetts, Amherst (USA) respectively. Together with Dr. Adrian Cristal, he co-manages the Computer Architecture for Parallel Paradigms research group at BSC. His current research interests include many-core computer architecture, reliability, low-power computing, programming models and transactional memory.

Ken Mai received his BS, MS, and PhD degrees in electrical engineering from Stanford University in 1993, 1997, and 2005, respectively. He joined the Faculty of Carnegie Mellon University in 2005 as an Assistant Professor in the Electrical and Computer Engineering Department. His research interests are in high-performance circuit design, secure IC design, reconfigurable computing, and computer architecture. He was the recipient of an NSF CAREER award in 2007 and the George Tallman Ladd Research Award in 2008.