Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

Limitations on MLC Flash Page Reuse and its Effects on Durability

Gala Yadgar*, Alexander Yucovich*, Hila Arobas*, Eitan Yaakobi*,
Yue Li§, Fabio Margaglia†, André Brinkmann† and Assaf Schuster*

*Computer Science Department, Technion
§California Institute of Technology
†Johannes Gutenberg-Universität Mainz

Abstract

Flash memory is prevalent in modern servers and devices. Coupled with the scaling down of flash technology, the popularity of flash memory motivates the search for methods to increase flash reliability and lifetime. Erasures are the dominant cause of flash cell wear, but reducing them is challenging because flash is a *write-once* medium— memory cells must be erased prior to writing.

An approach that has recently received considerable attention relies on *write-once memory (WOM)* codes, designed to accommodate additional writes on write-once media. However, most techniques proposed for reusing flash pages with WOM codes are limited to SLC flash, and are not applicable to MLC flash due to the specific constraints it imposes.

In this study, we use a hardware evaluation platform to identify the limitations of reprogramming flash pages on state-of-the-art MLC flash chips, and to directly measure the short and long-term effects of page reuse on SSD durability and energy consumption. Our findings provide key guidelines for FTL designs that employ WOM codes for reusing flash pages.

1 Introduction

Flash memories have special characteristics that make them especially useful for solid-state drives (SSD). Their short read and write latencies and increasing throughput provide a great performance improvement compared to traditional harddisk based drives. However, once a flash cell is written upon, changing its value from 1 to 0, it must be erased before it can be rewritten. In addition to the latency they incur, these erasures wear the cells, degrading their reliability. Thus, flash cells have a limited lifetime, measured as the number of erasures a block can endure.

Multi-level flash cells (MLC), which support four voltage levels, increase available capacity but have especially short lifetimes, as low as several thousands of erasures. Many methods for reducing block erasures have been suggested for incorporation in the flash translation layer (FTL), the SSD management firmware. These include minimizing user and internal write traffic [8, 11, 12, 19, 25, 26, 30, 32, 36] and distributing erasure costs evenly across the drive's blocks [3, 14, 16, 18].

A promising technique for reducing block erasures is to use write-once memory (WOM) codes. WOM codes alter the logical data before it is physically written, thus allowing the reuse of cells for multiple writes. They ensure that, on every consecutive write, ones may be overwritten with zeros, but not vice versa. Reusing flash cells with this technique might make it possible to increase the amount of data written to the block before it must be erased.

Flash page reuse is an appealing approach because it is orthogonal to other FTL optimizations. Indeed, the design of WOM codes and systems that use them has received much attention in recent years. While the coding theory community focuses on optimizing these codes to reduce their redundancy and complexity [4, 5, 7, 9, 31, 34], the storage community focuses on SSD designs that can offset these overheads and be applied to real systems [15, 24, 35].

However, the application of WOM codes to state-of-the-art flash chips is not straightforward. MLC chips impose additional constraints on modifying their voltage levels. Previous studies that examined page reuse on real hardware identified some limitations on reprogramming MLC flash, and thus resort to page reuse only on SLC flash [15], outside an SSD framework [10], or within a limited special-purpose FTL [20].

Thus, previous SSD designs that utilize WOM codes have not been implemented on real platforms, and their benefits were analyzed by simulation alone, raising the concern that they could not be achieved in real world storage systems. In particular, hardware aspects such as possible increase in cell wear and energy consumption due to the additional writes and higher resulting voltage levels

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

have not been examined before, but may have dramatic implications on the applicability of this approach.

In this study, we present a low-level evaluation of four state-of-the-art MLC flash chips. We examine the possibility of several reprogramming schemes for MLC flash and their short and long-term effects on the chip's durability, as well as the difference in energy consumption compared to that of traditional use.

Our key findings are as follows:

- Page reuse in MLC flash is possible, but can utilize only half of the pages and only if some of its capacity has been reserved in advance.
- The portion of the block's lifetime in which its pages can be reused safely depends on the characteristics of its chip.
- With WOM encoded data, the energy consumed by the additional flash operations is larger than that required by the saved erase operations.

The discrepancy between our results and previous ones emphasizes why understanding low-level constraints on page reuse is crucial for high-level designs and their objectives. We present the lessons learned from our analysis in the form of guidelines to be taken into account in future FTL designs, implementations, and optimizations.

The rest of this paper is organized as follows. Section 2 describes the basic concepts that determine the limitations on flash page reuse and its effects. We describe our experimental setup in Section 3 and identify the limitations on page reuse in MLC flash in Section 4. We evaluate the effects of page reuse on durability in Section 5, and its implications on energy consumption in Section 6. We survey related work in Section 7, and conclude in Section 8.

2 Preliminaries

In this section, we introduce the basic concepts of WOM codes and MLC flash that motivate our study.

2.1 Write-Once Memory Codes

Write-once memory (WOM) codes were first introduced in 1982 by Rivest and Shamir, for recording information multiple times on a write-once storage medium [28]. They give a simple WOM code example, presented in Table 1. This code enables the recording of two bits of information in three cells twice, ensuring that in both writes the cells change their value only from 1 to 0. For example, if the first message to be stored is 00, then 110 is written, programming only the last cell. If the second message is 10, then 010 is written, programming the first cell as well. Note that without special encoding, 00 cannot be overwritten by 10 without prior erasure. If the first and second messages are identical, then the cells do not change their value between the first and second writes. Thus, before performing a second write, the cell values must be *read* in order to determine the correct encoding.

Data bits	1st write	2nd write
11	111	000
01	011	100
10	101	010
00	110	001

Table 1: WOM code example

WOM code instances, or *constructions*, differ in the number of achievable writes and in the manner in which each successive write is encoded. The applicability of a WOM code construction to storage depends on three characteristics: (a) the *capacity overhead* —the number of extra cells required to encode the original message, (b) the encoding and decoding *efficiency*, and (c) the *success rate*—the probability of producing an encoded output that can be used for overwriting the chosen cells. Any two of these characteristics can be optimized at the cost of compromising the third.

Consider, for example, the code depicted in Table 1, where encoding and decoding are done by a simple table lookup, and therefore have complexity $O(1)$ and a success rate of 100%. However, this code incurs a capacity overhead of 50% on each write. This means that (1) only $\frac{2}{3}$ of the overall physical capacity can be utilized for logical data, and (2) every read and write must access 50% more cells than what is required by the logical data size.

The theoretical lower bound on capacity overhead for two writes is 29% [28]. Codes that incur this minimal overhead (*capacity achieving*) are not suitable for real systems. They either have exponential and thus inapplicable complexity, or complexity of $n \log n$ (where n is the number of encoded bits) but a failure rate that approaches 1 [5, 37].

Two recently suggested WOM code families, Polar [4, 5] and LDPC [37], have the same complexities as the error correction codes they are derived from. For these complexities, different constructions incur different capacity overheads, and the failure rate decreases as the capacity overhead increases. Of particular interest are constructions in which the overhead of the first write is 0, i.e., one logical page is written on one physical page. The data encoded for the second write requires two full physical pages for one logical page. Such a construction is used in the designs of ReusableSSD [35] and LLH-FTL [21].

We focus our evaluation on these WOM code constructions because they require minimal architectural support: the physical and logical pages do not have to be modified, and only full pages are programmed. We discuss the applicability of our findings to the general case in the following sections.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

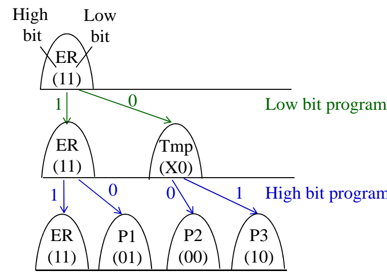


Figure 1: Normal programming order and states of MLC flash. ER is the initial (erased) state.

2.2 Multi-Level Cell (MLC) Flash

A flash memory chip is built from floating-gate cells whose state depends on the number of electrons they retain. Writing is done by *programming* the cell, increasing the *threshold voltage* (V_{th}) required to activate it. Cells are organized in blocks, which are the unit of erasure. Blocks are further divided into pages, which are the read and program units.

Single-level cells (SLC) support two voltage levels, mapped to either 1 (in the initial state) or 0. Thus, SLC flash is a classic write-once memory, where pages can be reused by programming some of their 1's to 0's. We refer to programming without prior erasure as *reprogramming*. Multi-level cells (MLC) support four voltage levels, mapped to 11 (in the initial state), 01, 00 or 10. This mapping, in which a single bit is flipped between successive states, minimizes bit errors if the cell's voltage level is disturbed. The least and most significant bits represented by the voltage levels of a multi-level cell are mapped to two separate pages, the *low page* and *high page*, respectively. These pages can be programmed and read independently. However, programming must be done in a certain order to ensure that all possible bit combinations can be read correctly. Triple-level cells (TLC) support eight voltage levels, and can thus store three bits. Their mapping schemes and programming constraints are similar to those of MLC flash. We focus our discussion on MLC flash, which is the most common technology in SSDs today.

Figure 1 depicts a normal programming order of the low and high bits in a multi-level cell. The cell's initial state is the erased (ER) state corresponding to 11. The low bit is programmed first: programming 1 leaves the cell in the erased state, while programming 0 raises its level and moves it to a temporary state. Programming the high bit changes the cell's state according to the state it was in after the low bit was programmed, as shown in the bottom part of the figure.[1] We discuss the implications of this mapping scheme on page reuse in the following section.

Bit errors occur when the state of the cell changes unintentionally, causing a bit value to flip. The reliability of a flash block is measured by its *bit error rate (BER)*—the average number of bit errors per page. The high voltage applied to flash cells during repeated program and erase operations gradually degrades their ability to retain the applied voltage level. This causes the BER to increase as the block approaches the end of its lifetime, which is measured in program/erase (P/E) cycles.

Bit errors in MLC flash are due mainly to *retention errors* and *program disturbance* [6]. Retention errors occur when the cell's voltage level gradually decreases below the boundaries of the state it was programmed to. Program disturbance occurs when a cell's state is altered during programming of cells in a neighboring page. In the following section, we discuss how program disturbance limits MLC page reuse, and evaluate the effects of reusing a block's pages on its BER.

Error correction codes (ECC) are used to correct some of the errors described above. The redundant bits of the ECC are stored in each page's *spare area*. The number of bit errors an ECC can correct increases with the number of redundant bits, chosen according to the expected BER at the end of a block's lifetime [37].

2.3 Flash Translation Layer

Write requests cannot update the data in the same place it is stored, because the pages must first be erased. Thus, writes are performed *out-of-place*: the previous data location is marked as invalid, and the data is written again on a clean page. The *flash translation layer (FTL)* is the SSD firmware component responsible for mapping logical addresses to physical pages.

When WOM codes are employed for reusing flash pages, the FTL is also responsible for determining whether a logical page is written in a first or a second write, and for recording the required metadata. For example, in some previous designs the FTL uses second writes only for hot data [24, 35]. Others rely on the host to indicate that pages are rewrite compatible [17, 20].

In our evaluation, we assume such an FTL would run on the SSD controller, and utilize the physical page and block operations supported by the flash controller. Thus, when a logical page is written as a second write, the invalidated data on the physical pages to be overwritten is first read into the SSD controller. The old data is used as input for the WOM encoder, which ensures that the new logical data can indeed be written on these pages without prior erasure.

3 Flash Evaluation Setup

Flash chips do not support reprogramming via their standard interfaces. Thus, the implications of reprogramming on the cells' state transitions and durability cannot be derived from standard documentation, and require experi-

[1] Partially programming the high bit in the temporary state is designed to reduce program disturbance.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

	A16	A27	B16	B29	C35
Feature size	16nm	27nm	16nm	29nm	35nm
Page size	16KB	8KB	16KB	4KB	8KB
Pages/block	256	256	512	256	128
Spare area (%)	10.15	7.81	11.42	5.47	3.12
Lifetime	3K	5K	10K	10K	NA

Table 2: Evaluated flash chip characteristics. A, B and C represent different manufacturers. The lifetime is the minimum between that expected and that observed in our experiments.

mentation with specialized hardware. We performed a series of experiments with several state-of-the-art flash chips to evaluate the limitations on reprogramming MLC flash pages and the implications of reprogramming on the chip's lifetime, reliability, and energy consumption.

We used four NAND flash chips from two manufacturers and various feature sizes, detailed in Table 2. We also include in our discussion the observations from a previous study on a chip from a third manufacturer [20]. Thus, our analysis covers three out of four existing flash manufacturers.

Chip datasheets include the expected lifetime of the chip, which is usually the maximal number of P/E cycles that can be performed before the average BER reaches 10^{-3}. However, cycling the chips in a lab setup usually wears the cells faster than normal operation because they program and erase the same block continuously. Thus, the threshold BER is reached after fewer P/E cycles than expected. In our evaluation, we consider the lifetime (T) of the chips as the minimum of the expected number of cycles, and the number required to reach a BER of 10^{-3}.

Our experiments were conducted using the SigNASII commercial NAND flash tester [1]. The tester allows software control of the physically programmed flash blocks and pages within them. By disabling the ECC hardware we were able to examine the state of each cell, and to count the bit errors in each page.

Some manufacturers employ *scrambling* within their chip, where a random vector is added to the logical data before it is programmed. Scrambling achieves uniform distribution of the flash cell levels, thus reducing various disturbance effects. In order to control the exact data that is programmed on each page, we bypass the scrambling mechanism on the chips that employ it.

Our evaluation excludes retention errors, which occur when considerable time passes between programming and reading the blocks. Reprogramming might increase the probability of retention errors because it increases the cell's V_{th}. However, since reprogramming is intended primarily for hot data, we believe it will not cause additional retention errors. Techniques such as "baking" [2] the chips in order to speed up the occurrence of retention errors are outside the scope of our evaluation.

4 Limitations on reprogramming

Flash cell reprogramming is strictly limited by the constraint that V_{th} can only increase, unless the block is erased. At the same time, WOM encoding ensures that reprogramming only attempts to change the value of each bit from 1 to 0. However, additional limitations are imposed by the scheme used for mapping voltage levels to bit values, and by the need to avoid additional program disturbance. Thus, page reuse must follow a *reprogramming scheme* which ensures that all reprogrammed cells reach their desired state.

We use our evaluation setup to examine which state transitions are possible in practice. We first consider three reprogramming schemes in which a block has been fully programmed, and show why they are impractical. We then validate the applicability of reprogramming when only the low pages of the block have been programmed before. Note that the state transitions do not depend on a specific WOM code or construction. Rather, they represent the requirement that bit values can only change from 1 to 0. Thus, the limitations identified in this section apply to the general case of page reprogramming with WOM codes.

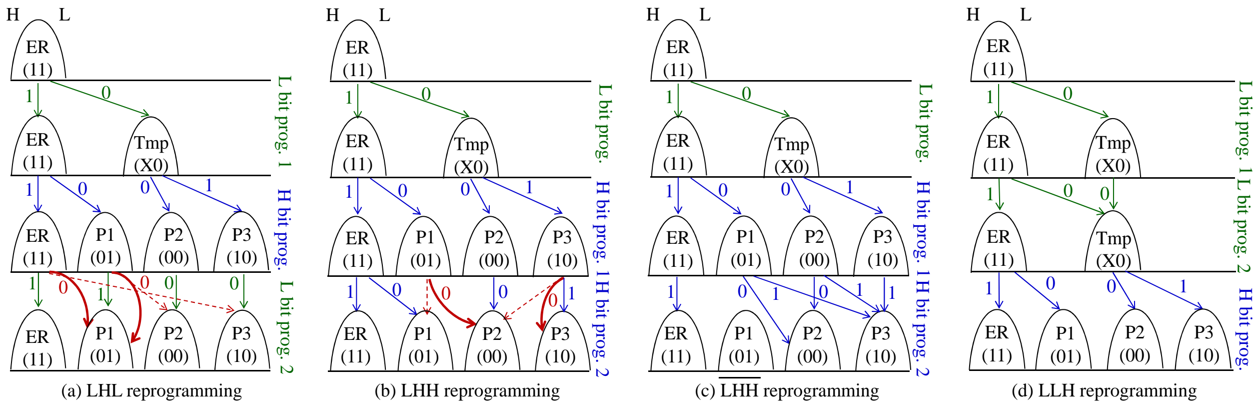
Let us assume that the entire block's pages have been programmed before they are reused. Thus, the states of the cells are as depicted in the bottom row of Figure 1. In the *low-high-low (LHL)* reprogramming scheme, depicted in Figure 2(a), we attempt to program the low bit from this state. The thin arrows depict possible desired transitions in this scheme. Two such transitions are impossible, resulting in an undesired state (depicted by the bold arrow). In the *low-high-high (LHH)* reprogramming scheme, depicted in Figure 2(b), the high page is reprogrammed in a fully used block. Here, too, two state transitions fail.

A possible reason for the failed transitions in the LHL scheme is that the voltage applied by the command to program the low bit is not high enough to raise V_{th} from $P1$ to $P2$ and from ER to $P3$.[2] The transition from $P3$ to $P2$ in the LHH scheme is impossible, because it entails decreasing V_{th}. Another problem in the LHH scheme occurs in state $P1$ when we attempt to leave the already programmed high bit untouched. Due to an unknown disturbance, the cell transitions unintentionally to $P2$, corrupting the data on the corresponding low page.

Three of these problematic transitions can probably be made possible with proper manufacturer support—the transition from $P3$ to $P2$ in the LHH scheme would be possible with a different mapping of voltage levels to states, and the two transitions in the LHL scheme

[2]We note that the transition from ER to $P3$ actually succeeded in the older, C35 chip [20]. All other problematic transitions discussed in this section failed in all the chips in Table 2.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) LHL reprogramming (b) LHH reprogramming (c) $\overline{\text{LHH}}$ reprogramming (d) LLH reprogramming

Figure 2: State transitions in the three reprogramming schemes. A thin arrow represents an attempted transition. A dashed arrow represents a failed transition, with a bold arrow representing the erroneous transition that takes place instead. Only LLH reprogramming achieves all the required transitions for page reuse without program disturbance.

could succeed if a higher voltage was applied during reprogramming. However, applying such architectural changes to existing MLC flash technology might amplify program disturbance and increase the BER. Thus, they require careful investigation and optimization.

An alternative to modifying the state mapping is modifying the WOM encoding, so that the requirement that 1's are only overwritten by 0's is replaced by the requirement that 0's are only overwritten by 1's. Figure 2(c) shows the resulting *$\overline{\text{low-high-high}}$ ($\overline{\text{LHH}}$)* reprogramming scheme. Its first drawback is that it corrupts the low pages, so a high page can be reused only if the data on the low page is either invalid, or copied elsewhere prior to reprogramming. Such reprogramming also corrupted the high pages adjacent to the reprogrammed one. Thus, this scheme allows safe reprogramming of only one out of two high pages. The benefits from such a scheme are marginal, as these pages must also store the redundancy of the encoded data.

Interestingly, reprogramming the high bits in chips from manufacturer A returned an error code and did not change their state, regardless of the attempted transition. A possible explanation is that this manufacturer might block reprogramming of the high bit by some internal mechanism to prevent the corruption described above.

The problems with the LHL and LHH schemes motivated the introduction of the *low-low-high (LLH)* reprogramming scheme by Margaglia et al. [20]. Blocks in this scheme are programmed in two rounds. In the first round only the low pages are programmed. The second round takes place after most of the low pages have been invalidated. All the pages in the block are programmed in order, i.e., a low page is reprogrammed and then the corresponding high page is programmed for the first time, before moving on to the next pair of pages.

We validated the applicability of the LLH scheme on the chips of manufacturers A and B. Figure 2(d) depicts the corresponding state transitions of the cells. Since both programming and reprogramming of the low bit leave the cell in either the erased or temporary state, there are no limitations on the programming of the high page in the bottom row. This scheme works well in all the chips we examined. However, it has the obvious drawback of leaving half of the block's capacity unused in the first round. This leads to the first lesson from our low-level evaluation.

Lesson 1: *Page reuse in MLC flash is possible, but can utilize only half of the pages and only if some of its capacity has been reserved in advance. FTL designs must consider the implications of this reservation.*

5 Flash Reliability

In analyzing the effects of reprogramming on a chip's durability, we distinguish between *short-term* effects on the BER due to modifications in the current P/E cycle, and *long-term* wear on the cell, which might increase the probability of errors in future cycles. With this distinction, we wish to identify a *safe* portion of the chip's lifetime, during which the resulting BER as well as the long term wear are kept at an acceptable level.

5.1 Average V_{th} and BER

Reprogramming increases the probability that a cell's value is 0. Thus, the average V_{th} of reused pages is higher than that of pages that have only been programmed once. A higher V_{th} increases the probability of a bit error. The short-term effects of increased V_{th} include increased program disturbance and retention errors, which are a direct result of the current V_{th} of the cell and its neighboring cells. The long-term wear is due to the higher voltage applied during programming and erasure.

Our first set of experiments evaluated the short-term effects of increased V_{th} on a block's BER. In each chip,

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

Num. of P_{LLH} cycles	A16	A27	B16	B29
T (= entire lifetime)	32%	29%	20%	30.5%
$0.6 \times T$	8%	9%	8%	9%
$0.4 \times T$	6%	6.5%	6%	6.5%
$0.2 \times T$	2%	3%	3%	3.5%

Table 3: Expected reduction in lifetime due to increased V_{th}.

Num. of LLH cycles	A16	A27	B16	B29
T (= entire lifetime)	38%	59.5%	99%	31%
$0.6 \times T$	8.5%	8%	7%	8.5%
$0.4 \times T$	5.2%	6%	5%	5.5%
$0.2 \times T$	1%	2.5%	3%	3%

Table 4: Expected reduction in lifetime due to reprogramming.

we performed T regular P/E cycles writing random data on one block, where T is the lifetime of the chip as detailed in Table 2. We repeated this process on different blocks, with different distributions of 1 and 0. $P_{0.5}$, in which the probability of a bit to be 0 is 0.5, is our baseline. With P_{LLH} the probability of 0 was 0.75 and 0.5 in the low and high page, respectively. This corresponds to the expected probabilities after LLH reprogramming.

Manufacturing variations may cause different planes on the same chip to exhibit slightly different behavior. Thus, we repeated each experiment on six blocks, three in each plane. We read the block's content and recorded the BER after every P/E cycle. We then calculated the average for each plane and compare the result to the baseline on that plane. The difference between the planes was minor (less than 1%) in all our experiments. Thus, in the following, we present the overall average results for all six blocks in each experiment.

The implication of an increase in BER depends on whether it remains within the error correction capabilities of the ECC. A small increase in BER at the end of a block's lifetime might deem it unusable, while a large increase in a 'young' block has little practical effect. For a chip with lifetime T, let T' be the number of cycles required to reach a BER of 10^{-3} in this experiment. Then $T - T'$ is the *lifetime reduction* caused by increasing V_{th}.

Our results, summarized in Table 3, were consistent in all the chips we examined. Programming with P_{LLH}, which corresponds to a higher average V_{th}, shortened the chips' lifetime considerably, by 20–32%.

In the next set of experiments, we evaluated the long-term effects of V_{th}. Each experiment had two parts: we programmed the block with P_{LLH} in the first part, for a portion of its lifetime, and with $P_{0.5}$ in the second part, which consists of the remaining cycles. Thus, the BER in the second part represents the long-term effect of the biased programming in the first part. We varied the length of the first part between 20%, 40% and 60% of the block's lifetime. Figure 3 shows the BER in the different parts of each experiment, with the lifetime reduction summarized in Table 3.

Our results show that the long term effect of increasing V_{th} is modest, though nonnegligible—increasing V_{th} early in the block's lifetime shortened it by as much as 3.5%, 6.5% and 9%, with increased V_{th} during 20%, 40% and 60% of the block's lifetime, respectively.

P_{LLH} is the probability of a bit to be 0 in the specific WOM code constructions described in Section 2.1. While this probability may vary for different codes and constructions, it is always higher than 0.5. Thus, some increase in cell wear should be expected whenever flash pages are reused with WOM codes, although the extent of this effect may vary according to the WOM code in use.

5.2 Reprogramming and BER

In the third set of experiments, we measured the effects of reprogramming by performing T LLH reprogramming cycles on blocks in each chip. Figure 4 shows the BER results, and Table 4 summarizes the expected lifetime reduction.

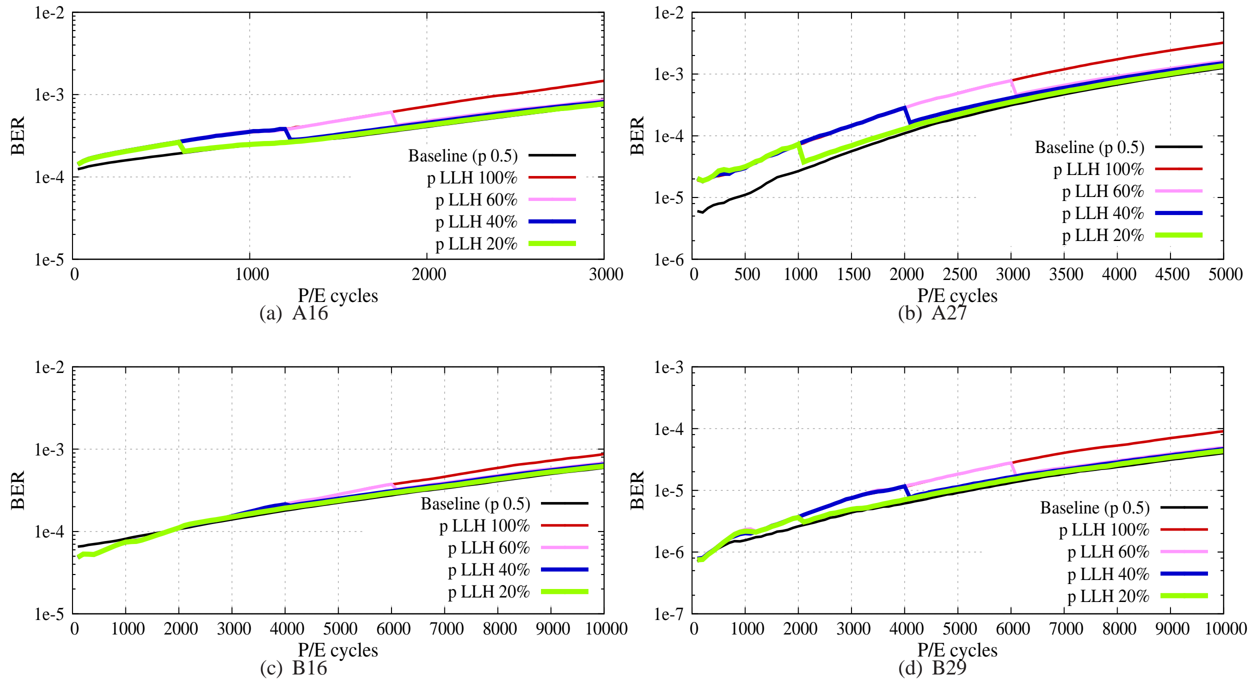
In all but one chip, the BER in the first round of programming the low pages was extremely low, thanks to the lack of interference from the high pages. In the second round, however, the BER of all pages was higher than the baseline, and resulted in a reduction of lifetime greater than that caused by increasing V_{th}. We believe that a major cause of this difference are optimizations specifically tailored for the regular LH programming order [27]. These optimizations are more common in recent chips, such as the B16 chip.

In some chips, the programming of the low page is performed with a low degree of accuracy, relying on the subsequent programming of the high page to bring the cell to its final state. We believe that such optimization can explain the results of the B29 chip (Figure 4(d)). On this chip, in the first 30% of the block's lifetime, the first round of programming the low pages (L1) resulted in a BER which was higher than that of the baseline. This explanation is supported by the second round of programming: During this part of the block's lifetime, the BER of the low page after reprogramming (L2) was very close to that of the baseline.

In the last set of experiments, we evaluated the long term effects of reprogramming. Here, too, each experiment was composed of two parts: we programmed the block with LLH reprogramming in the first part, and with $P_{0.5}$ and regular programming in the second part. We varied the length of the first part between 20%, 40% and 60% of the block's lifetime. Figure 5 shows the BER results, and Table 4 summarizes the expected lifetime reduction.

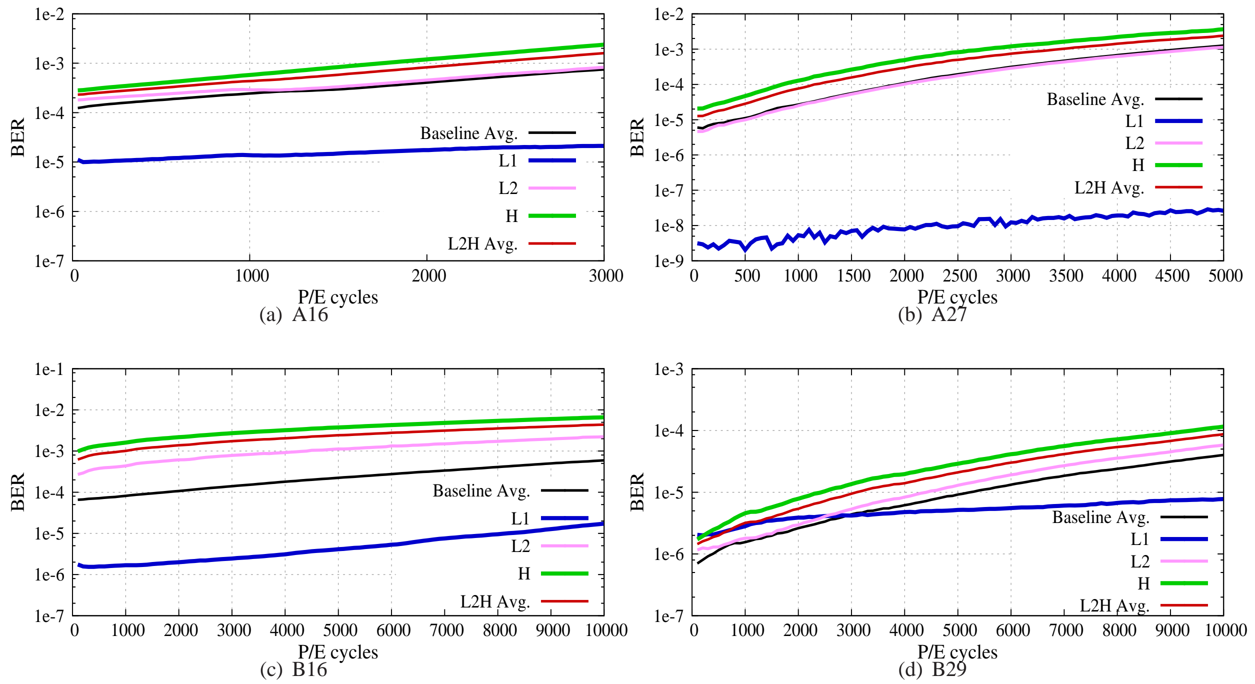
We observe that the long-term effects of reprogram-

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) A16 (b) A27 (c) B16 (d) B29

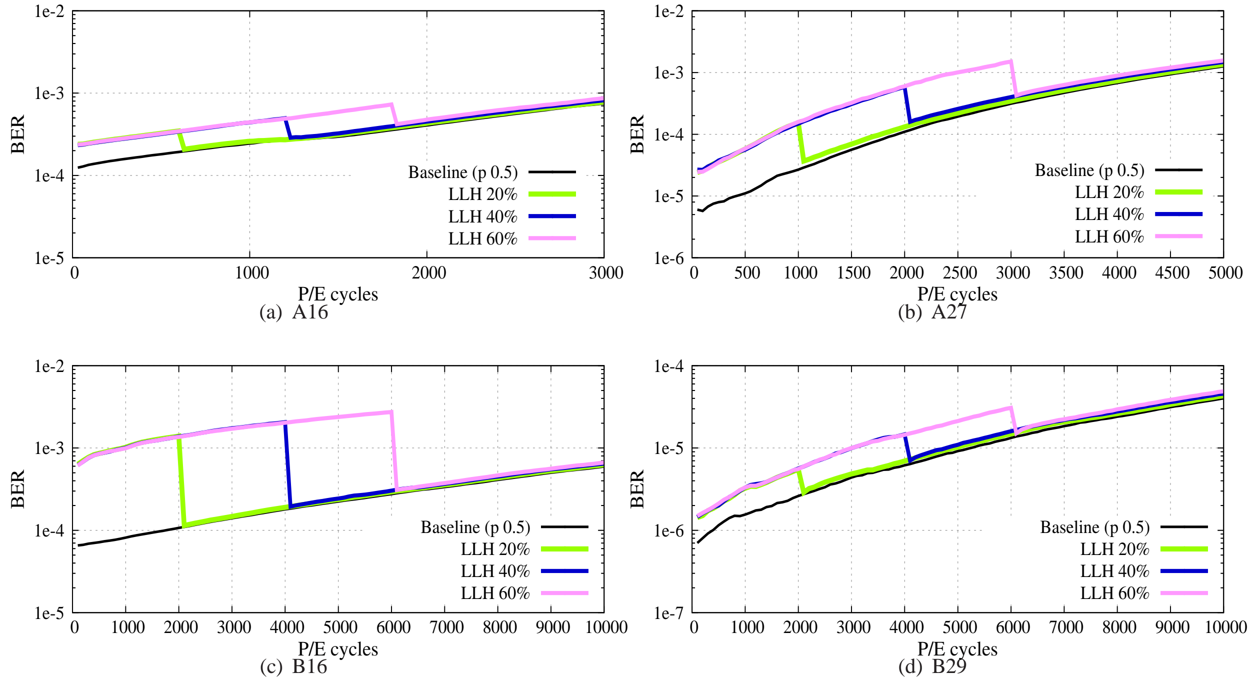
Figure 3: Short and long term effects of increased V_{th} on each chip.



(a) A16 (b) A27 (c) B16 (d) B29

Figure 4: Short term effects of reprogramming on each chip.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) A16 (b) A27 (c) B16 (d) B29

Figure 5: Long term effects of reprogramming on each chip.

ming are modest, and comparable to the long term effects of increasing V_{th}. This supports our assumption that the additional short-term increase in BER observed in the previous set of experiments is not a result of the actual reprogramming process, but rather of the mismatch between the programming order the chips are optimized for and the LLH reprogramming scheme. This is especially evident in the B16 chip, in which the BER during the first part was high above the limit of 10^{-3}, but substantially smaller in the second part of the experiment.

Thus, schemes that reuse flash pages only at the beginning of the block's lifetime can increase its utilization without degrading its long-term reliability. Moreover, in all but the B16 chips, LLH reprogramming in the first 40% of the block's lifetime resulted in BER that was well within the error correction capabilities of the ECC. The designs of ReusableSSD [35] and LLH-FTL [21] are based on similar assumptions.

We note, however, that the variance between the chips we examined is high, and that short and long-term effects do not depend only on the feature size. For example, the A16 chip is "better" than the A27 chip, but the B16 chip is "worse" than the B29 chip. This leads to the second lesson from our low-level evaluation.

Lesson 2: *The portion of the block's lifetime in which its pages can be reused safely depends on the characteristics of its chip. The FTL must take into account the long term implications of reuse on the chips it is designed for.*

Operation	Baseline (μJ)	LLH (μJ)
Erase	192.79	186.49
Read (L)	50.37	50.37
Read (H)	51.25	51.25
Program (L_1)	68.18	68.55
Reprogram (L_2)	NA	63.04
Program (H)	195.65	180.85
Average logical read	50.81	60.79
Average logical write	132.64	145.71

Table 5: Energy consumed by flash operations on chip A16.

6 Energy consumption

Flash read, write and erase operations consume different amounts of energy, which also depend on whether the operation is performed on the high page or on the low one, and on its data pattern. We examined the effect of reprogramming on energy consumption by connecting an oscilloscope to the SigNAS tester. We calculated the energy consumed by each of the following operations on the A16 chip: an erasure of a block programmed with P_{LLH} and p=0.5, reading and writing a high and a low page, reprogramming a low page, and programming a high page on a partially used block.

To account for the transfer overhead of WOM encoded data, our measurements of read, program and reprogram operations included the I/O transfer to/from the registers. Our results, averaged over three independent measure-

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

ments, are summarized in Table 5. We also present the average energy consumption per read or write operation with baseline and with LLH reprogramming, taking into account the size of the programmed data, the reading of used pages for supplying the invalid data as input to the WOM encoder, and the number of pages that can be written before each erasure.

These results show that page reuse consumes more overall energy than the baseline. This is in contrast to previous studies showing possible energy savings. These studies assumed that the energy is proportional to the number of programmed *cells*, which is equivalent in a first and in a second write [10, 35]. However, our hardware evaluation shows that the number of reprogrammed *pages* is the dominant factor in energy consumption. While reprogramming a lower page consumes less energy than the average logical write in the baseline, the use of WOM encoding entails an extra read and page reprogram for each logical write. The low energy consumption of the saved erase operations does not offset the additional energy consumed by those operations. We note, however, that when page reuse reduces the internal writes by the FTL, some energy savings may result.

The measurements we performed were specific to the WOM code constructions in Section 2.1—they consider programming one physical page for a first write, and reading and reprogramming two physical pages for a second write. While these WOM codes are not optimal in terms of storage overhead, they require the minimal number of flash operations on existing flash architectures. Alternative codes that require encoding the data of the first write also require more than one physical page to be programmed. At the same time, even if the second write requires less than two full physical pages, two pages must still be reprogrammed. Since our results show that the number of flash operations is the dominant factor in energy consumption, they serve as a lower bound on the energy consumption of flash page reuse with WOM codes in general. Thus, we draw the following lesson from these results.

Lesson 3: *With WOM encoded data, the energy consumed by the additional flash operations is larger than that required by the saved erase operations. Energy savings are possible only if they reduce the number of write operations performed on the flash chip.*

7 Related Work

Several studies proposed FTL designs that reuse pages to extend SSD lifetime. Some are based on capacity achieving codes, and bound the resulting capacity loss by limiting second writes to several blocks [24] or by assuming the logical data has been compressed by the upper level [15]. The overheads and complexities in these designs are addressed in the design of ReusableSSD [35]. However, none of these studies addressed the limitations of reprogramming MLC flash pages. Some of these limitations were addressed in the design of an overwrite compatible B$^+$-tree data structure, assuming the mapping of V_{th} to bits can be modified [17]. Like the previous approaches, it has been implemented only in simulation. Extended P/E cycles [20] were implemented on real hardware, but the FTL that uses them relies on the host to supply and indicate data that is overwrite compatible. LLH-FTL [21], which was designed based on the observations in this study, is the first general-purpose FTL that addresses all practical limitations of WOM codes as well as MLC flash. Thus, it demonstrate its strengths and weaknesses on real hardware and workloads.

Numerous studies explored the contributors to BER in flash, on a wide variety of chip technologies and manufacturers. They show the effects of erasures, retention, program disturbance and scaling down technology on the BER [6, 10, 22, 33]. These studies demonstrate a trend of increased BER as flash feature sizes scale down, and the need for specialized optimizations employed by manufacturers as a result. Thus, we believe that some of the interference effects observed in our experiments are a result of optimizing the chips for regular LH programming. Adjusting these optimizations to LLH reprogramming is a potential approach to increase the benefit from page reuse.

Several studies examined the possibility of reprogramming flash cells. Most used either SLC chips [15], or MLC chips as if they were SLC [9]. A thorough study on 50nm and 72nm MLC chips demonstrated that after a full use of the block (LH programming), half of the pages are “WOM-safe” [10]. However, they do not present the exact reprogramming scheme, nor the problems encountered when using other schemes. A recent study [20] mapped all possible state transitions with reprogramming on a 35nm MLC chip, and proposed the LLH reprogramming scheme. Our results in Section 5 show that smaller feature sizes impose additional restrictions on reprogramming, but that LLH reprogramming is still possible.

Previous studies examined the energy consumption of flash chips as a factor of the programmed pattern and page [23], and suggested methods for reducing the energy consumption of the flash device [29]. To the best of our knowledge, this study is the first to measure the effect of reprogramming on the energy consumption of a real flash chip and incorporate it into the evaluation of the FTL.

Recent trends in flash technologies, such as one-shot programming and 3D V-NAND [13], eliminate the constraints on the programming order of pages in each block. This may allow reprogramming pages on a fully used block, and maybe even allow reprogramming of the low

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

and high pages alike. To understand their implications, these technologies should be evaluated separately.

8 Conclusions

We presented the first comprehensive study of the effects of reusing flash pages with WOM codes on real flash chips. We showed that page reuse in MLC flash is possible, but can utilize only half of the pages and only if some of its capacity has been reserved in advance. While reprogramming is safe for at least 40% of the lifetime of the chips we examined, it incurs additional *long-term* wear on their blocks.

A reduction in erasures does not necessarily translate to a reduction in energy consumption, which is determined by the overall amount of physical pages read and written. The reduction in physical flash page writes is limited by the storage overhead of WOM encoded data, and is mainly constrained by the limitation of reusing only half of the block's pages.

This study exposed a considerable gap between the previously shown benefits of page reuse, which were based on theoretical analysis and simulations, and those that can be achieved on current state-of-the-art hardware. This gap should be carefully addressed by the design of future FTLs. At the same time, we believe that most of the limitations on these benefits can be addressed with manufacturer support, and that the potential benefits of page reuse justify reevaluation of current MLC programming constraints.

References

[1] NAND flash memory tester (SigNASII). http://www.siglead.com/eng/innovation_signas2.html, 2014.

[2] Solid state drive (SSD) requirements and endurance test method. Standard, JEDEC, 2014.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *USENIX Annual Technical Conference (ATC)*, 2008.

[4] D. Burshtein. Coding for asymmetric side information channels with applications to polar codes. In *IEEE International Symposium on Information Theory (ISIT)*, 2015.

[5] D. Burshtein and A. Strugatski. Polar write once memory codes. *IEEE Transactions on Information Theory*, 59(8):5088–5101, 2013.

[6] Y. Cai, O. Mutlu, E. Haratsch, and K. Mai. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *31st IEEE International Conference onComputer Design (ICCD)*, 2013.

[7] G. D. Cohen, P. Godlewski, and F. Merkx. Linear binary code for write-once memories. *IEEE Transactions on Information Theory*, 32(5):697–700, 1986.

[8] J. Colgrove, J. D. Davis, J. Hayes, E. L. Miller, C. Sandvig, R. Sears, A. Tamches, N. Vachharajani, and F. Wang. Purity: Building fast, highly-available enterprise flash storage from commodity components. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2015.

[9] E. En Gad, H. W., Y. Li, and J. Bruck. Rewriting flash memories by message passing. In *IEEE International Symposium on Information Theory (ISIT)*, 2015.

[10] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[11] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam. Leveraging value locality in optimizing NAND flash-based SSDs. In *9th USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[12] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng. Improving flash-based disk cache with lazy adaptive replacement. In *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, 2013.

[13] J.-W. Im et al. A 128Gb 3b/cell V-NAND flash memory with 1gb/s i/o rate. In *IEEE International Solid- State Circuits Conference (ISSCC)*, 2015.

[14] S. Im and D. Shin. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *J. Syst. Archit.*, 56(12):641–653, Dec. 2010.

[15] A. Jagmohan, M. Franceschini, and L. Lastras. Write amplification reduction in NAND flash through multi-write coding. In *26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[16] X. Jimenez, D. Novo, and P. Ienne. Wear unleveling: Improving NAND flash lifetime by balancing page endurance. In *12th USENIX Conference on File and Storage Technologies (FAST)*, 2014.

[17] J. Kaiser, F. Margaglia, and A. Brinkmann. Extending SSD lifetime in database applications with page overwrites. In *6th International Systems and Storage Conference (SYSTOR)*, 2013.

[18] T. Kgil, D. Roberts, and T. Mudge. Improving

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

NAND flash based disk caches. In *35th Annual International Symposium on Computer Architecture (ISCA)*, 2008.

[19] H. Kim and S. Ahn. BPLRU: A buffer management scheme for improving random writes in flash storage. In *6th USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[20] F. Margaglia and A. Brinkmann. Improving MLC flash performance and endurance with extended P/E cycles. In *IEEE 31st Symposium on Mass Storage Systems and Technologies (MSST)*, 2015.

[21] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann. The devil is in the details: Implementing flash page reuse with WOM codes. In *14th USENIX Conference on File and Storage Technologies FAST*, 2016.

[22] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill. Bit error rate in NAND flash memories. In *Reliability Physics Symposium (IRPS). IEEE International*, 2008.

[23] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. Stan, and S. Swanson. Modeling power consumption of NAND flash memories using FlashPower. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(7):1031–1044, July 2013.

[24] S. Odeh and Y. Cassuto. NAND flash architectures reducing write amplification through multi-write codes. In *IEEE 30th Symposium on Mass Storage Systems and Technologies (MSST)*, 2014.

[25] Y. Oh, J. Choi, D. Lee, and S. H. Noh. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *10th USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[26] H. Park, J. Kim, J. Choi, D. Lee, and S. Noh. Incremental redundancy to reduce data retention errors in flash-based SSDs. In *IEEE 31st Symposium on Mass Storage Systems and Technologies (MSST)*, 2015.

[27] K.-T. Park, M. Kang, D. Kim, S.-W. Hwang, B. Y. Choi, Y.-T. Lee, C. Kim, and K. Kim. A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories. *IEEE Journal of Solid-State Circuits*, 43(4):919–928, April 2008.

[28] R. L. Rivest and A. Shamir. How to Reuse a Write-Once Memory. *Inform. and Contr.*, 55(1-3):1–19, Dec. 1982.

[29] M. Salajegheh, Y. Wang, K. Fu, A. Jiang, and E. Learned-Miller. Exploiting half-wits: Smarter storage for low-power devices. In *9th USENIX Conference on File and Stroage Technologies (FAST)*, 2011.

[30] M. Saxena, M. M. Swift, and Y. Zhang. FlashTier: A lightweight, consistent and durable storage cache. In *7th ACM European Conference on Computer Systems (EuroSys)*, 2012.

[31] A. Shpilka. Capacity achieving multiwrite WOM codes. *IEEE Transactions on Information Theory*, 60(3):1481–1487, 2014.

[32] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending SSD lifetimes with disk-based write caches. In *8th USENIX Conference on File and Storage Technologies (FAST)*, 2010.

[33] E. Yaakobi, L. Grupp, P. Siegel, S. Swanson, and J. Wolf. Characterization and error-correcting codes for TLC flash memories. In *International Conference on Computing, Networking and Communications (ICNC)*, 2012.

[34] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf. Codes for write-once memories. *IEEE Transactions on Information Theory*, 58(9):5985–5999, 2012.

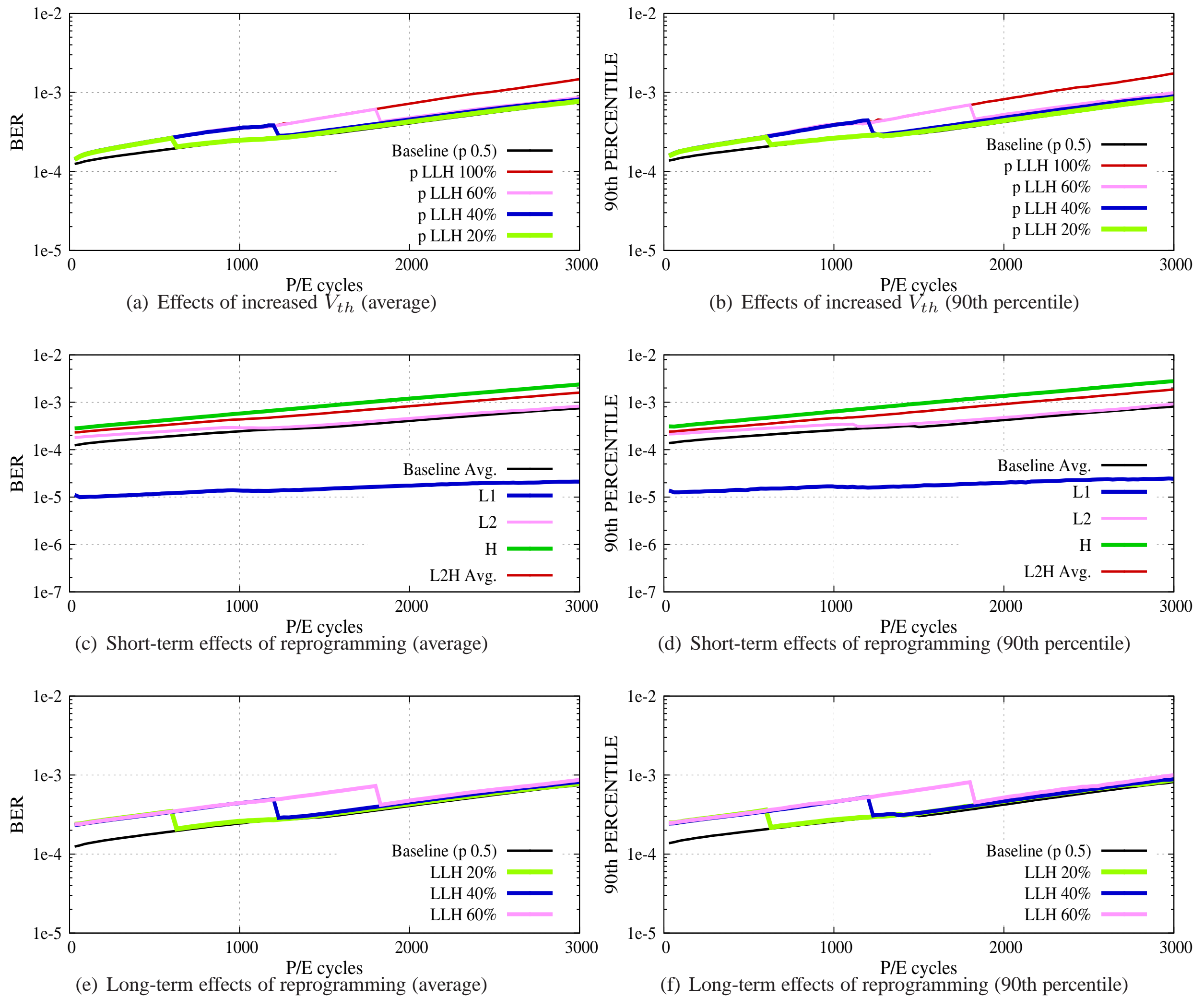
[35] G. Yadgar, E. Yaakobi, and A. Schuster. Write once, get 50% free: Saving SSD erase costs using WOM codes. In *13th USENIX Conference on File and Storage Technologies (FAST)*, 2015.

[36] J. Yang, N. Plasson, G. Gillis, and N. Talagala. HEC: Improving endurance of high performance flash-based cache devices. In *6th International Systems and Storage Conference (SYSTOR)*, 2013.

[37] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang. LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives. In *11th USENIX Conference on File and Storage Technologies (FAST)*, 2013.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016

Appendix: Results by Chip



(a) Effects of increased V_{th} (average)

(b) Effects of increased V_{th} (90th percentile)

(c) Short-term effects of reprogramming (average)

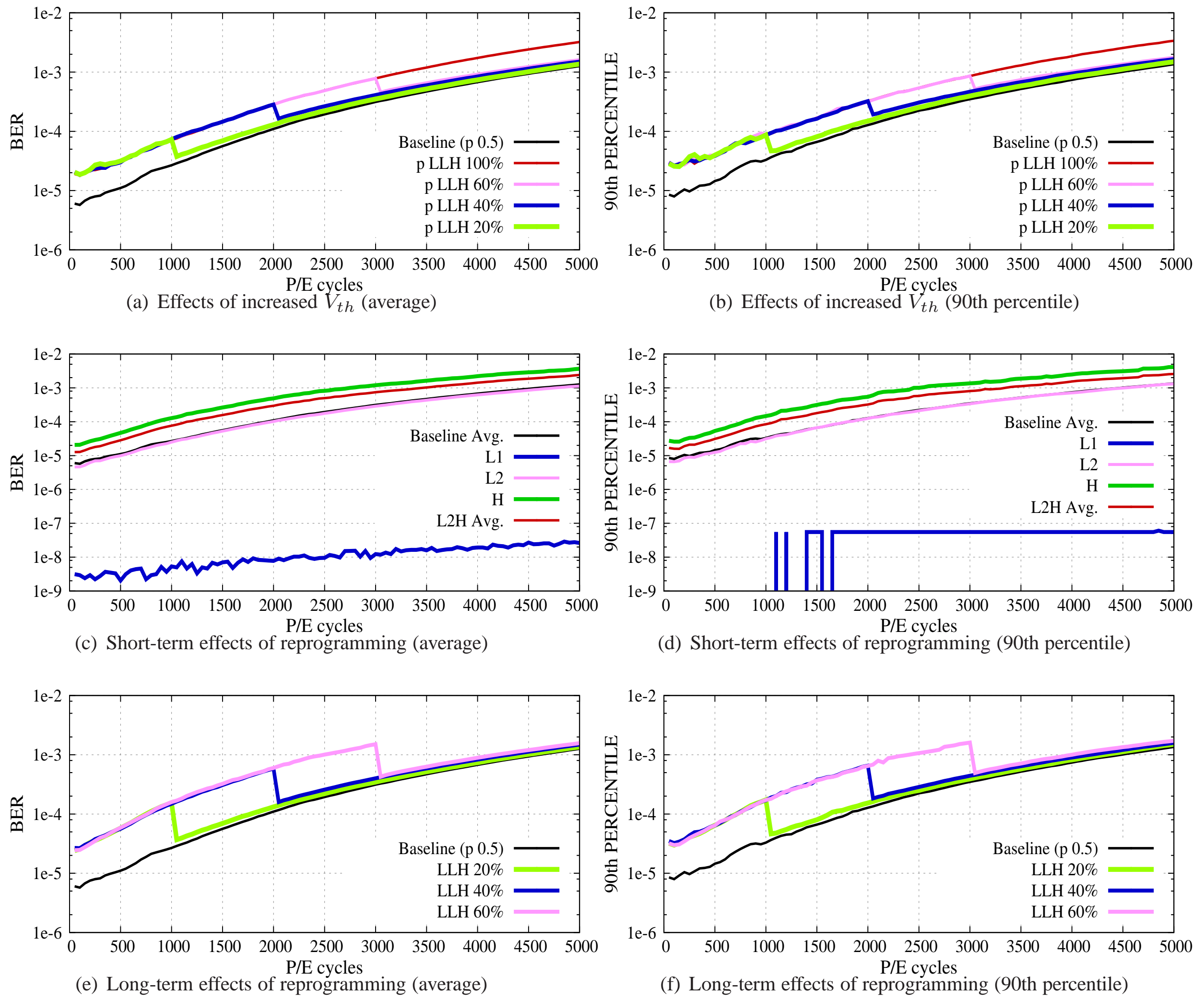
(d) Short-term effects of reprogramming (90th percentile)

(e) Long-term effects of reprogramming (average)

(f) Long-term effects of reprogramming (90th percentile)

Figure 6: Average and 90th percentile BER of the A16 chip in all experiments.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) Effects of increased V_{th} (average)

(b) Effects of increased V_{th} (90th percentile)

(c) Short-term effects of reprogramming (average)

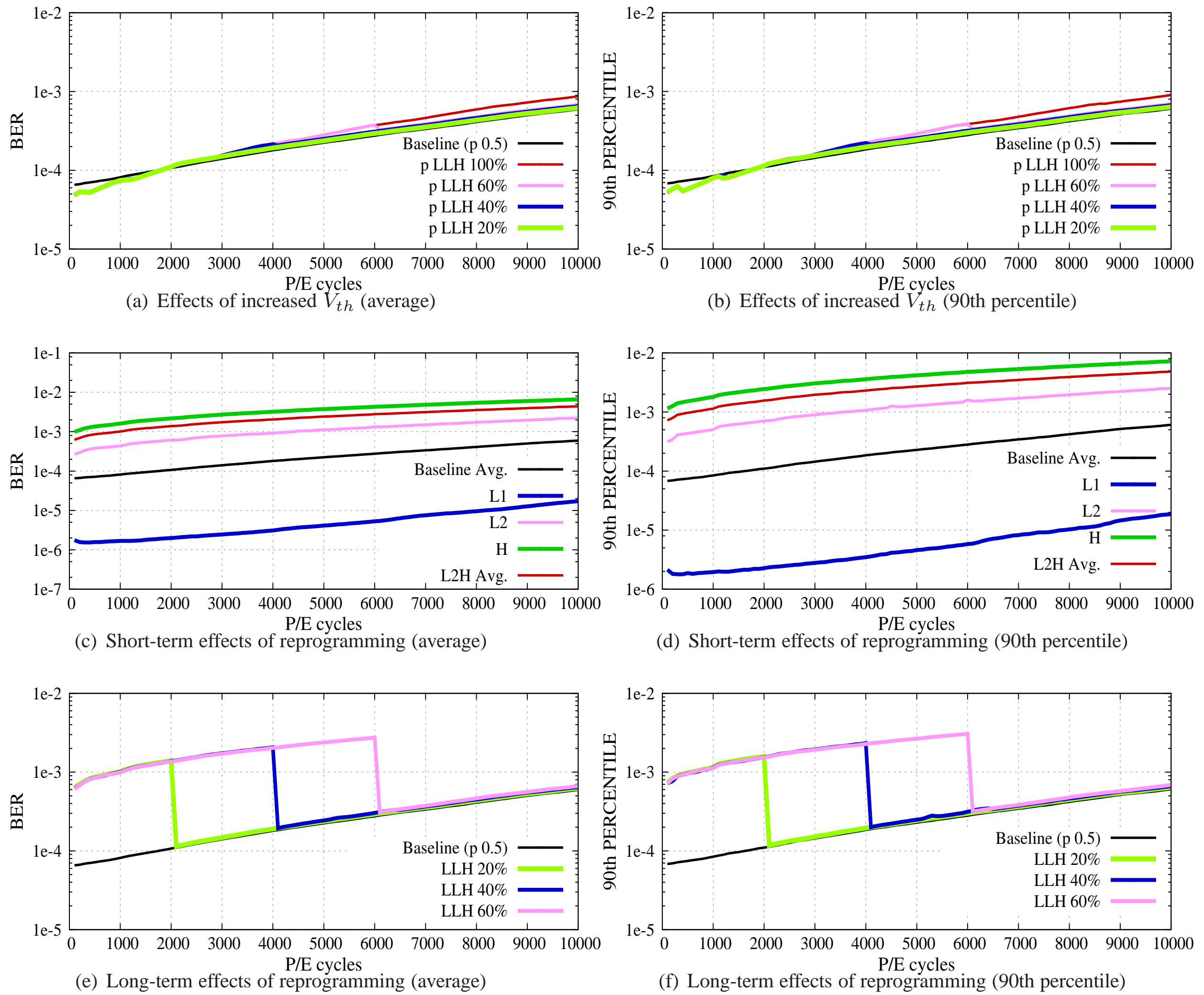
(d) Short-term effects of reprogramming (90th percentile)

(e) Long-term effects of reprogramming (average)

(f) Long-term effects of reprogramming (90th percentile)

Figure 7: Average and 90th percentile BER of the A27 chip in all experiments.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) Effects of increased V_{th} (average)

(b) Effects of increased V_{th} (90th percentile)

(c) Short-term effects of reprogramming (average)

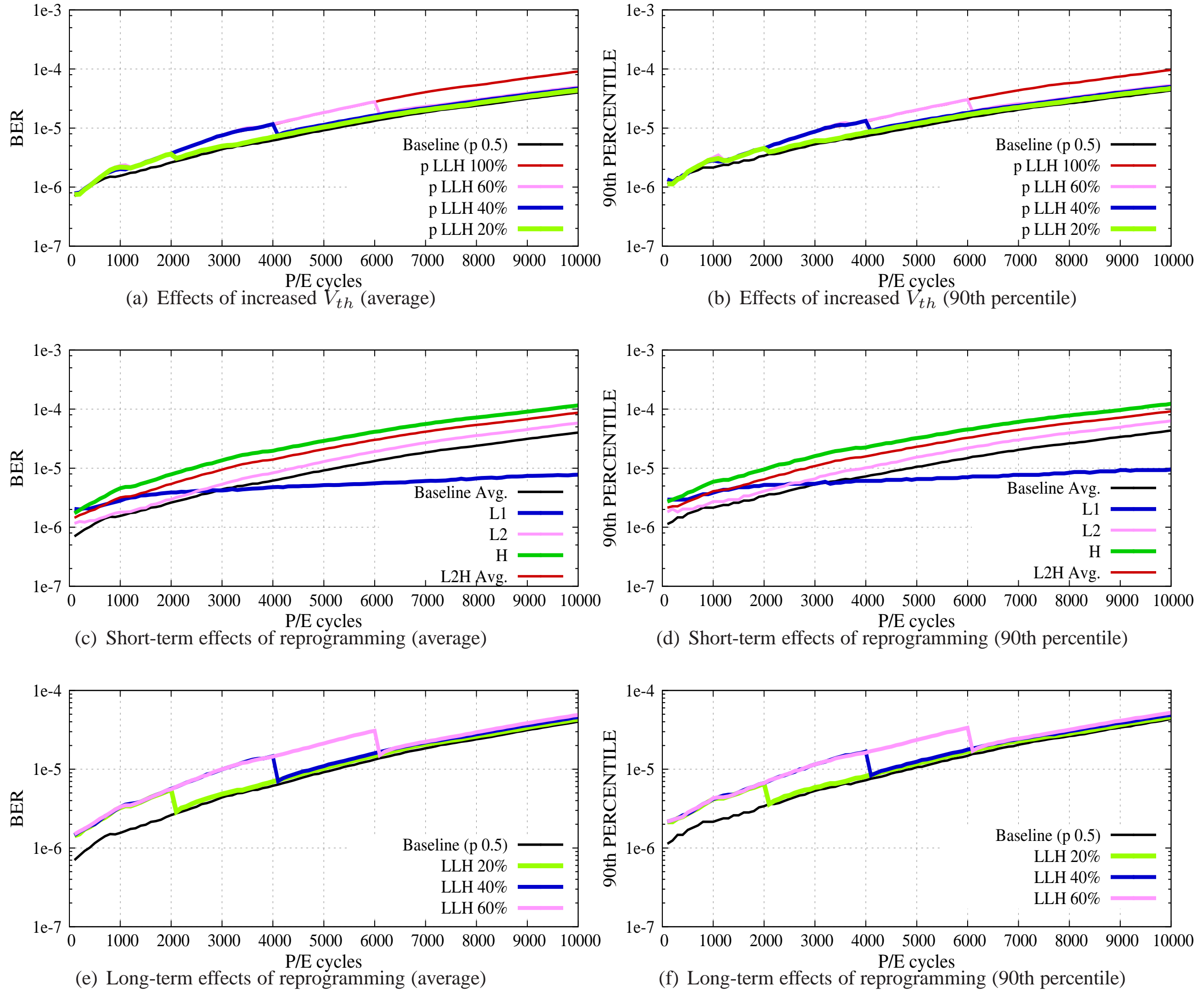
(d) Short-term effects of reprogramming (90th percentile)

(e) Long-term effects of reprogramming (average)

(f) Long-term effects of reprogramming (90th percentile)

Figure 8: Average and 90th percentile BER of the B16 chip in all experiments.

Technion - Computer Science Department - Tehnical Report CS-2016-02 - 2016



(a) Effects of increased V_{th} (average)

(b) Effects of increased V_{th} (90th percentile)

(c) Short-term effects of reprogramming (average)

(d) Short-term effects of reprogramming (90th percentile)

(e) Long-term effects of reprogramming (average)

(f) Long-term effects of reprogramming (90th percentile)

Figure 9: Average and 90th percentile BER of the B29 chip in all experiments.