Improving MLC flash performance and endurance with Extended P/E Cycles

Fabio Margaglia
Johannes Gutenberg-Universität
Mainz, Germany
Email: margagl@uni-mainz.de

André Brinkmann
Johannes Gutenberg-Universität
Mainz, Germany
Email: brinkman@uni-mainz.de

***Abstract*—The traditional usage pattern for NAND flash memory is the *program/erase* (P/E) cycle: the *flash pages* that make a *flash block* are all programmed in order and then the whole flash block needs to be erased before the pages can be programmed again. The erase operations are slow, wear out the medium, and require costly *garbage collection* procedures. Reducing their number is therefore beneficial both in terms of performance and endurance. The physical structure of flash cells limits the number of opportunities to overcome the 1 to 1 ratio between programming and erasing pages: a bit storing a logical 0 cannot be reprogrammed to a logical 1 before the end of the P/E cycle.**

This paper presents a technique to minimize the number of erase operations called *extended P/E cycle*. With extended P/E cycles, the flash pages can be programmed many times before the whole flash block needs to be erased, reducing the number of erase operations. We study the applicability of the technique to *Multi Level Cell* (MLC) NAND flash chips, and present a design and implementation on the OpenSSD prototyping board. The evaluation of our prototype shows that this technique can achieve erase operations reduction as high as 85%, with latency speedups of up to 67%, with respect to a FTL with traditional P/E cycles, and naive greedy garbage collection strategy. Our evaluation leads to valuable insights on how extended P/E cycles can be exploited by future applications.

I. Introduction

NAND flash memory is currently the solid state technology most widely used in storage systems, mainly packaged as Solid State Drives (SSD). It offers great read and write performance, and a low power consumption. Despite these substantial improvements relative to hard disk drives (HDD), the gap in cost per gigabyte still limits the adoption of SSDs as replacements for HDDs. Flash manufacturers try to drive down costs by storing more bits per flash cell to increase its density, introducing Multi Level Cell (MLC) flash. This technology uses cells with 4 or 8 states (respectively 2 or 3 bits per cell), as opposed to Single Level Cell (SLC) flash, which has 2 states (1 bit per cell). In this paper we focus on MLC chips with 4 states which are the most common in both consumer and enterprise SSDs.

Every state is characterized by a threshold voltage (V_{th}). When more bits are packed in a flash cell the V_{th} margins between different states are reduced and the processes required to detect the difference need to be more precise, and consequently become slower and more prone to errors. Therefore, while flash density increases rapidly, all other critical metrics – performance, endurance, and data retention – decrease at a higher pace. The metric showing the steepest and most alarming decline is the endurance. A recent study on 45 flash chips shows that each additional bit per flash cell reduces the expected lifetime by up to 20x [1].

The major cause of the limited lifetime in flash technology is the cell wear out caused by the *erase operation*. This operation is required to bring the flash cells to their initial state, which corresponds to the lowest V_{th}. Since a program operation can only increase the V_{th} of a cell, the cell must be erased before being programmed again, so that the cell can be set to an arbitrary state. Hence, the traditional usage of flash memory in the program/erase cycles (P/E cycles). An approach to reduce the erase operations is that of reprogramming the flash bits without erasing them first, sacrificing the possibility to reach states with lower V_{th}. In this paper we refer to this technique as *extended P/E cycles* because it allows multiple program operations per cycle. Theoretical studies and simulations of this approach have been presented in [2]–[4].

Performing multiple program operations per cycle has the constraint that the reprogrammed cells cannot be set to an arbitrary value. The previous work in this area is based on the assumption that flash cells can be safely programmed multiple times as long as the bits are flipped in a single direction (e.g. 1→0). In this paper we refer to this assumption as the *write-once memory constraint* (*WOM constraint*), and to data that complies with it as *WOM compatible*. The WOM constraint was based on the behavior of punched cards and optical disks, but applies to today's flash SLC chips as well because they have only two states. Previous work shows how to achieve WOM compatible data patterns through encodings [5], or specialized data structures [3].

However, the WOM constraint can not be applied directly to MLC flash, because every cell is characterized by 4 states, encoded into 2 bits. Therefore, when a bit is reprogrammed, the state of the flash cell is modified, possibly corrupting the value of the other bit. This phenomenon happens also when the bits are flipped according to the WOM constraint, and is called *program disturbance* [6].

In this paper we address the applicability of extended P/E cycles to MLC flash. Our main contribution is the identification and definition of the additional constraints imposed by MLC technology, which we refer to as the *practical MLC WOM constraints*. We present these constraints in a state diagram.

978-1-4673-7619-8/15/$31.00 ©2015 IEEE

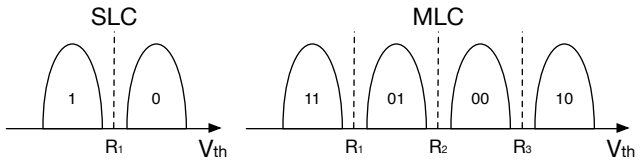

Fig. 1. Distributions of threshold voltages for SLC and MLC. Each distribution shows the logical bits that it represents. R_0, R_1, R_2, R_3 are the read points. Due to wear out the V_{th} distributions change over the lifetime of a cell. A cell becomes unreadable when the distributions cross the read points.

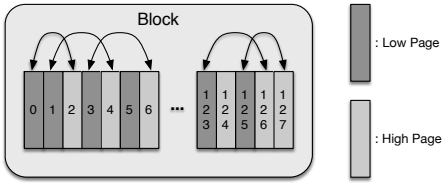

Fig. 2. Low and High pages in a block. The arrows show the couples that share the same flash cells and are thus influenced when programming each other.

While WOM compatible MLC reprogramming has been discussed in [6], to the best of our knowledge this is the first presentation of such a diagram with detailed information of all possible state transitions. We believe that this tool will be useful to system designers and coding theorists for further optimizations of the flash utilization. In this work we chose to use only the WOM compatible state transitions to design a Flash Translation Layer (FTL) that uses extended P/E cycles without sacrificing flash capacity. All previous FTL designs were based on SLC flash [2], or on simulations that did not consider the MLC constraints [4], [7]. We implemented and evaluated our FTL design using the OpenSSD platform [8], which is an experimental kit for SSD design and development equipped with 35 nm MLC chips.

The rest of this paper is structured as follows: Section II introduces the main concepts concerning flash and FTLs, and gives the background on WOM codes and WOM-compatible data structures. Section III presents the state diagram for a typical MLC flash cell. Section IV describes the FTL implementation, and Section V presents its evaluation. Section VI presents the related work, and Section VII concludes.

II. Background

A. Flash Operations

A flash chip consists of an array of cells. Each cell is a transistor with a floating gate that can retain electrons. The cell's state depends on the number of electrons in the floating gate because they hinder the normal behavior of the transistor, increasing the threshold voltage (V_{th}) required to activate it. Therefore, a cell's state can be determined by probing the voltage required to activate the transistor.

The floating gate's electron population can be altered with two operations: the flash *program* operation injects new electrons, while the *erase* operation depletes the floating gate completely. V_{th} can be probed with the *read* operation. The cells are read and programmed in *pages* of size ranging from 8 KB to 32 KB, and are erased in *blocks* of 2 MB to 4 MB.

Program operations can only increase the cell's state, therefore any attempt to reprogram a page will fail if a cell's state should decrease as a result of the operation. In the general case, in-place page updates are prohibited, and the pages must be erased and then programmed again, hence the traditional *Program/Erase cycles* (P/E) usage pattern.

B. SLC and MLC

Depending on the precision of the machinery necessary to probe the threshold voltage, it is possible to distinguish between more states per flash cell. As shown in Figure 1, Single Level Cells (SLCs) distinguish between 2 states, while Multi Level Cells (MLCs) distinguish between 4 states. Therefore, each SLC represents 1 bit of information, while each MLC represents 2 bits.

Every MLC has 4 levels encoded into 2 bits. The two bits of a MLC are mapped to two independently accessible logical pages, called *low page* and *high page*. Therefore, when either of these two pages is programmed, the V_{th} of the cell increases, changing the cell's state and possibly the value of the other bit. However, the flash manufacturers guarantee that if the low page is programmed first, and the high page later, the two operations will be successful and will not influence one another.

Figure 2 shows the configuration of low and high pages in the chips we analyzed. Grupp et al. presented an analysis of different chips in [6] that shows that all the chips they tested are configured in a similar way, if not identical. Note that, with this configuration, programming the pages in order implies that the low page is always programmed before the high page.

C. Flash Translation Layer

The Flash Translation Layer (FTL) is a dedicated firmware, that runs on the controller of the SSDs, that is responsible for managing the flash chips. It has to convert the host requests into flash operations, and is optimized to handle the lack of in-place updates, and the big size of erase blocks. It commonly structures the available flash memory as a log of pages. The in-place updates are avoided by appending the updated page to the log. A *mapping table* is maintained by the FTL to keep track of these page remappings.

The log structure requires garbage collection because the remapping of pages leaves stale copies that accumulate during the log lifetime. The garbage collection process scans the flash for a victim erase block, copies the valid pages contained therein to the end of the log, and erases the block. The victim block selection affects performance because it determines how much data has to be copied. The popular *greedy strategy* chooses the block with the lowest amount of valid data [9]. Other strategies have been proposed, which consider the hotness and age of the data [10].

D. OpenSSD

The platform we use to implement our prototype is the OpenSSD prototyping kit that hosts an Indilinx

TABLE I. TWO GENERATIONS WOM CODE

Plain bits	1^{st} gen	2^{nd} gen
00	111	000
01	110	001
10	101	010
11	011	100

Barefoot™controller, and eight slots for custom flash modules [8]. Each module hosts four 64 Gb 35 nm MLC flash chips, for a total of 32 GB of flash memory per module. The flash chips have a page size of 32 KB, and erase blocks made of 128 pages (4 MB). Every page has 512 extra bytes referred to as *spare region*. This region is meant to be used to store FTL metadata. However, in the OpenSSD board, the spare region is not directly accessible by the controller.

E. Bit Errors and ECC

During their lifetime, the flash cells are subject to wear-out. The repeated injections and removals of electrons in the floating gate ruin the physical structure of the cell, leading to its malfunction. This causes bit errors in the flash pages. Typically *error correction codes* (ECC) are used to improve the reliability of flash.

The OpenSSD platform is shipped with a hardware accelerator that transparently computes the ECC and stores it in the spare region at each flash program operation. This behavior can not be modified, and the spare region can not be accessed by the controller. This ECC can correct up to 12 B per 512 B segment. When the controller issues a flash page read operation the hardware accelerator corrects the bit errors on the fly during the operation. When the page contains more errors than can be corrected, the hardware accelerator raises an incorrectable error interrupt to the controller, and leaves the data untouched.

For the sake of the analysis presented in this paper it was important to circumvent the ECC to observe the raw behavior of the cells when using the extended P/E cycles. Unfortunately, due to limitations of the OpenSSD board, we could not disable the ECC protection when writing entire pages. We could, however, disable the ECC protection when performing partial page write operations of only 512 B. This limitation influenced the set up of two experiments presented in this work: the analysis of the reprogram operation in MLC (Section III), and the bit error rate evaluation (Section V-A).

F. Page reprogramming in SLC

SLCs map the low V_{th} state to a logical 1, and the high V_{th} state to a logical 0 (see Figure 1). Therefore, a page can be reprogrammed with new data if all the bits are flipped in the 1→0 direction, or are not modified, with respect to the bits already contained in the page. We call this the WOM constraint. The problem of updating data complying with the WOM constraint has been tackled in two ways:

a) WOM codes: WOM codes, introduced by Rivest et al. [5], offer redundant encodings organized in generations, where the number of generations is the number of WOM compliant updates allowed by the code. Table I shows a 2 generations WOM code scheme that encodes bit pairs into 3 bit code words. The 0s in each code word of a generation are a superset of the 0s of all the different code words in the previous generations. Therefore, every code word can always be updated to a different code word of the next generation respecting the WOM constraint.

b) WOM compatible data structures: some data structures can be modified to perform WOM compatible accesses. Kaiser et al. presented a B-Tree with log structured nodes with unsorted entries [3]. The nodes are initialized with all bits set to 1. The new entries are appended to the node, in the initialized area, generating only 1→0 bit flips. Bloom filters [11] flip bits in a single direction by construction and are WOM compatible without need of modifications.

G. Program disturbance in MLC

Program disturbance is defined as the involuntary flipping of one or more bits of a page, that happens as a consequence of a program operation on a different page. Previous experiments have shown that reprogramming MLC flash causes program disturbance. In particular, Grupp et al. showed that reprogramming the high page disturbs the bits of the respective low page. On the other hand, reprogramming the low page does not affect the high page [6].

III. PAGE REPROGRAMMING IN MLC

In order to better understand the implications of the reprogram operation in MLC flash chips, we extracted the cell's state diagram for a typical MLC, which is shown in Figure 3. To the best of our knowledge, this is the first time that this information is presented for MLCs. Previous publications [3], [12], and the flash manufacturers, show only a tabular representation of the mapping (Table II). Despite showing the same states, the tabular representation lacks the transitions between them. These transitions are fundamental to identify the program patterns leading to program disturbance and data corruption.

TABLE II. TRADITIONAL V_{th} MAPPING REPRESENTATION

V_{th}	LB	HB
Min	1	1
	1	0
	0	0
Max	0	1

The experiment we used to extract the diagram was performed on the OpenSSD board. The procedure consisted of taking a couple of connected low/high pages (see Figure 2) and traversing every state transition by programming a single bit belonging either to the low or the high page, and then reading back the content of both pages. In order to correctly detect the content after the transition, we needed to avoid any interference of the ECC hardware accelerator of the OpenSSD board. Therefore, we disabled it, and to do so we had to use the partial write operations of 512 B (the only ones allowed). This was not a problem because our goal was to program a single bit every time. Once we extracted the complete diagram, we verified it with full page write operations. Since the ECC could not be disabled on full page operations, we made sure to overload the ECC correction capacity by flipping more bits than it could correct. The verification with full page operations performed in this way confirmed the diagram.

Our first observation is that there are in fact 5 states rather than 4 because the order in which bits are programmed affects the behavior of the cell in future program operations. Consistently

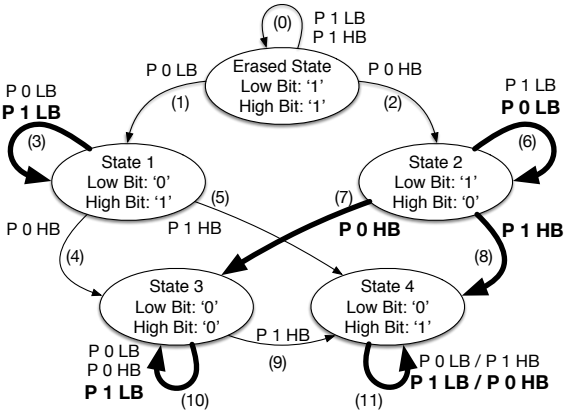

Fig. 3. States of a MLC flash cell. Each state is labeled with the corresponding bit values. Each transition represents a program operation either on the low bit or on the high bit. The highlighted transitions cause data corruption.

with SLC chips, the cells in the Erased State are mapped to 1 on both bits, and the bits are cleared in the states with higher V_{th}. Given this premise, ideally MLCs could be reprogrammed using the WOM constraint, according to which programming a 0 is always safe. Unfortunately, some transitions – highlighted in the diagram – break this assumption. In transition 6, the low bit remains at 1 even though it is programmed to 0, and in transition 11 this behavior is observed for the high bit. Furthermore, transition 7 causes program disturbance when programming the high bit to 0.

The use mode recommended by flash manufacturers accurately avoids these problematic transitions. According to the recommendation, the pages in the same erase block must be programmed in order and each page can be programmed only once. Because of the block organization shown in Figure 2, programming the pages in order implies that the low page is always programmed before the high page. The first program operation hits the low page and uses transition 0 or 1. The second program operation hits the high page and uses transitions 2, 4, or 5. All of these transitions are safe.

On the other hand, when we allow reprogramming operations on all pages of a block, we assume that either of the pages composing the pair, or both of them, might have been programmed before. Therefore, their cells can be in any state. In this case it becomes very difficult to avoid the problematic transitions.

Our second observation is that it is indeed possible to reprogram the low page, complying with the WOM constraint, before programming the high page. In this way only transitions 0, 1, and 3 are used (note that the WOM constraint guarantees that transition 3 is never used to program the low bit at 1). These transitions are all safe. When the low page has been reprogrammed it is still possible to program the high page one single time, using transitions 2, 4, or 5.

The combination of WOM compliant data, and this programming order ensures compatibility with the practical WOM constraints offered by MLC, effectively enabling the extended P/E cycles on MLC flash chips.

Based on these observations we propose a bimodal solution where every block can be used as a *write block* or as an *overwrite block*. A write block is used following the manufacturers' recommendations. An overwrite block is used according to our second observation in two steps. In the first step, only the low pages are used and reprogram operations are allowed. Then, in the second step, the program access to the low pages is forbidden, and the high pages are programmed in order one time. We call this second step the *block seal operation*. Essentially the sealed overwrite block becomes a write block where only the high pages can be used. The low pages can still be read and the data they contain will not be corrupted.

IV. FTL Implementation

In this section we present the FTLs we implemented on the OpenSSD board. The implementation is based on the design choice of leaving to the application the responsibility of deciding what data is WOM compliant. Therefore we augmented the SATA interface with the *SATA overwrite command*, which is used by the applications to write WOM compliant data. In our prototype this command is implemented by adding an extra bit to the logical block address (LBA) of the standard SATA write command, which is interpreted by the FTL as a switch between the write and the overwrite commands.

In the following we present the FTLs we implemented. The *baseline FTL* does not use the extended P/E cycles, and is used as the baseline for the evaluation. The *seal FTL* is a FTL which uses the extended P/E cycles and uses the *block seal operation* to handle the constraints of the overwrite blocks.

A. Baseline FTL

This FTL does not use the extended P/E cycles. Therefore whenever it receives a SATA overwrite command it considers it as a normal SATA write command. It uses a page-associative mapping table, which is large compared to mapping tables of other schemes (e.g. block associative, hybrid, and DFTL mapping schemes [13]–[15]). This scheme ensures maximal block utilization and has steady performance which we required as a reference point. However, the P/E cycle extension is orthogonal to the mapping granularity and can be applied to other mapping FTL schemes as well.

The flash pages are organized as a log, as described in Section II. The pages are updated out of place, appending the newer version to the end of the log, and updating the mapping table. The older copies of updated pages are reclaimed by garbage collection. The victim block to be cleaned is selected with the greedy strategy which always chooses the block with the lowest number of valid pages. An efficient priority queue keeps the full blocks ordered by the number of valid pages contained therein, and the FTL maintains one queue per bank. The victim block can therefore be found in constant time. Each full block has an *inverse mapping table* stored in its last page. This table stores the logical addresses of all the pages stored in the block which are used during garbage collection to index the FTL mapping table and find which pages still contain valid data.

The current prototype does not perform any kind of progressive background cleaning because the OpenSSD ARM controller has only one core that is busy managing the new requests: using it to perform background cleaning stalls the incoming

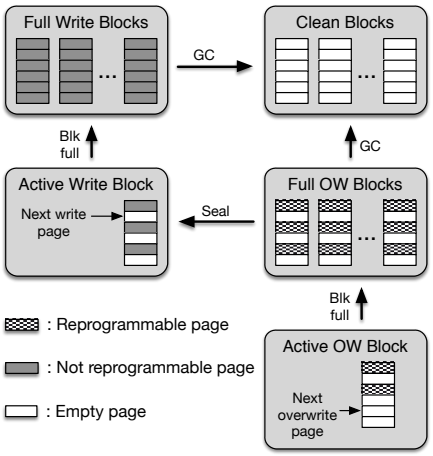

Fig. 4. Life cycle of blocks in the seal FTL. Clean blocks can be chosen as either write blocks or overwrite blocks. For every bank, the FTL maintains one active write block and one active overwrite block, with a pointer to the next free page. The low pages in all the overwrite blocks are reprogrammable until they are sealed. In the picture the active write block is an overwrite block which has just been sealed. The pointer to the next write page will point only to high pages. The garbage collection process can be performed both on full write blocks and full overwrite blocks.

request queue and harms performance. The only optimization we implemented consists of polling the priority queues of each idle bank in search of completely invalid blocks. These blocks can be erased with the asynchronous erase command offered by the flash chips while processing every SATA request. To implement this optimization the queue has been augmented with the ability to return the number of valid pages.

The garbage collection procedure is triggered whenever a bank contains only one clean block. The victim is chosen based on the priority queues described before. The garbage collection copies all the valid pages to the end of the log. We let the garbage collection copy valid data to other banks so that the whole routine is accelerated by parallelizing the program operations. The program operations are asynchronous, although the temporary DRAM buffers must be locked until the operation completes. The identification of the valid pages is sped up by the reverse mapping table stored in the last page of every full block.

The implementation of the garbage collection subroutine that copies the valid pages poses an interesting design choice. The flash chips offer the *copyback* operation, which is able to copy one entire page to a different page within the same flash bank. Since the data never leaves the bank, this operation is faster than reading a page and programming it to a new location. However, we did not employ the copyback operation for two reasons. First, we hide the cost of the program operation by running it asynchronously on the other banks. Second, when the copyback operation is used, the data never leaves the bank, therefore the ECC is neither checked on read nor recomputed on write. Any error will be copied to the new page weakening the overall protection.

B. Seal FTL

This FTL uses the extended P/E cycles, therefore it has been extended with the notion of overwrite blocks as described

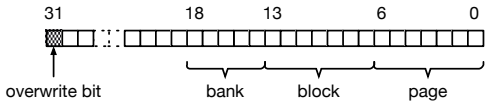

Fig. 5. Physical address stored in the mapping table for a SSD configured with 32 banks and 128 blocks per bank.

is Section III. Traditionally, the flash blocks exist in three states. They can be clean, full, or be the current active block where incoming data is written to. The seal FTL instead distinguishes between two main categories of flash blocks: write blocks and overwrite blocks. Therefore, blocks can exist in five different states: clean, full write block, full overwrite block, current active write block, and current active overwrite block (see Figure 4). In the full overwrite blocks and the active overwrite block only the low pages are used, therefore the page reprogram operation is allowed.

Overwrite Operation

For each request identified as a SATA overwrite command the FTL searches the corresponding physical addresses in the mapping table. The physical addresses, represented in Figure 5, identify a unique (bank, block, page) tuple. In addition, one bit is reserved for pages residing in full or active overwrite blocks (filled with a checkered pattern in Figure 4). When this bit is set the FTL can proceed reprogramming the page directly in its current physical page. If the bit is not set, or if it is the first time that the application addresses the page, the FTL uses the next low page in the active overwrite block.

The FTL maintains one reprogram counter per page in the DRAM. It is possible to set a limit to the reprogram operations per page. When a reprogram operation hits a page that reached the limit, the FTL uses the next low page in the active overwrite block instead, and decreases the number of valid pages for that block in the priority queue.

Overwrite Block Sealing

When the FTL needs a new block and there is only one block left in the clean pool, it can either trigger garbage collection or seal an overwrite block to use its high pages. We will discuss the empirical policy used to take this decision later, here we explain how the seal operation is implemented.

Once a block is sealed, its pages cannot be reprogrammed anymore. If a future overwrite operation hits one of the pages contained in the block, the FTL will be forced to find a new physical location for it, losing the overwrite potential. Therefore, the FTL attempts to select the block whose pages are least likely to be overwritten in the future. To do so, the FTL maintains a priority queue, identical to the one used for write blocks, to find the overwrite block with the lowest amount of valid pages in constant time. This block is chosen as a victim. The rationale behind this design is that, having less valid pages, the probability for a future overwrite hit is lower.

When the block is chosen, the FTL needs to clear all the overwrite bits in the corresponding physical addresses (see

Figure 5). To find them quickly the FTL keeps an inverse mapping table containing the logical addresses of all pages in the block. This inverse mapping table is stored in the last low page of the block, after all the other low pages are programmed. During the seal operation, the FTL reads the inverse mapping table and scans the FTL mapping table clearing all the overwrite bits of the pages that are still mapped to the block.

Finally, the FTL points the next write page pointer to the first high page in the block, and sets the flag that makes the next write page pointer increase in steps of 2 (according to the mapping shown in Figure 2). After this point the block is considered the active write block and its pages cannot be reprogrammed.

Garbage Collection

The garbage collection procedure is similar to the one used in the baseline FTL. Preferably the victims are full write blocks with a few valid pages, however the procedure works on full overwrite blocks, too, with the only difference that the inverse mapping table is located in the last low page, i.e. page 125 (see Figure 2), instead of the last high page. The inverse mapping table for full overwrite blocks has invalid entries corresponding to all the high pages, therefore the garbage collection procedure does not perform any operation on them. The following section explains when the garbage collection is triggered on full overwrite blocks.

Victim Selection Policies

The main tool for victim selection are the two priority queues for write and overwrite blocks. The preferred victims are full write blocks with few valid pages because they have been used completely, i.e. all the pages have been programmed, and the cleaning cost consists of copying the few valid pages they contain. In contrast, the full overwrite blocks should be left in that state for as long as possible, because their pages can be reprogrammed in the future.

However, if only the full write blocks are garbage collected, the full overwrite blocks accumulate, reducing the number of write blocks. Consequently, the garbage collection invocations become more frequent, and the full write blocks are reclaimed earlier, when they still contain a high number of valid pages.

To prevent this, the FTL can choose to seal a full overwrite block and use its high pages, instead of triggering a garbage collection. This choice is made when the victim in the overwrite priority queue has less valid pages than the victim in the write queue. This policy ensures that garbage collections do not have to copy too much data. An overwrite block can, in fact, have a maximum of 63 valid pages (the last low page is reserved for the inverse mapping table). Therefore, this is the upper bound for data copying during garbage collection.

The seal operation is not a solution when the FTL needs to allocate a new active overwrite block. This requires an erased block, and a garbage collection operation is inevitable. To prevent using garbage collection on full write blocks with many valid pages, the FTL can choose to select a full overwrite block as victim instead, if it has less valid pages. However, that means that the high pages are erased without ever being used. We have experimented with two strategies for victim selection:

- Comparing only the number of valid low pages in the overwrite blocks, and consider all the high pages (empty) as invalid.
- Comparing the number of valid low pages plus the number high pages.

The second option attempts to preserve the overwrite block from being reclaimed, taking into account the fact that the high pages could be written after an eventual seal operation. Section V presents an evaluation of both strategies.

ECC

The ECC hardware accelerator of the OpenSSD platform has a fixed behavior that can not be modified. When a page is programmed for the first time, the ECC is computed and stored in a fixed location of the spare area of the flash page. When the page is reprogrammed, a new ECC is computed and stored in the same location of the spare area. Since the ECC is not WOM compliant, this results in a corrupt ECC.

The OpenSSD platform does not offer any possibility to modify this behavior. Therefore, our prototype does not offer data protection on reprogrammed pages. However, we envision two possible solutions to this issue:

- Encode the data with a WOM code which offers error correction capabilities [16].
- Use the spare region as append-only log, and append newer ECCs after the old ones.

In our evaluation the FTL enforced a per page limit of 8 overwrites, to simulate 8 ECCs appended in the spare region, e.g. 8 ECCs of 64 B can be appended in the 512 B spare region offered by the flash chips shipped with the OpenSSD. Note that we did not disable the ECC accelerator when running the evaluation, therefore the ECC computation still happens, and its delay is considered in our experiments. The FTL ignores the ECC failure interrupts that occur when reading reprogrammed pages.

V. Evaluation

A. Impact of Extended P/E Cycles on Lifetime

In this section, we evaluate how the extended P/E cycles technique impacts the lifetime of a flash cell. The lifetime is expressed as the number of P/E cycles that a cell can sustain before becoming unusable. The *bit error rate* (BER) grows with repeated P/E cycles, to the point where the errors found in a page are too many to be corrected with ECCs. At that point the page can no longer be used safely.

The traditional BER analysis to evaluate the lifetime of a cell consists of the repetition of P/E cycles on a single flash block. During the program, phase every page in the block is programmed. During the erase phase, the entire block is erased. The content of the block is checked for errors after every phase, and the number of errors is used to compute the BER. This analysis has been performed by different research groups on different flash chips [1], [2], [6], [17].

To evaluate the effect of the extended P/E cycles on the flash lifetime, we modified the traditional BER analysis to program the pages multiple times in each program phase.

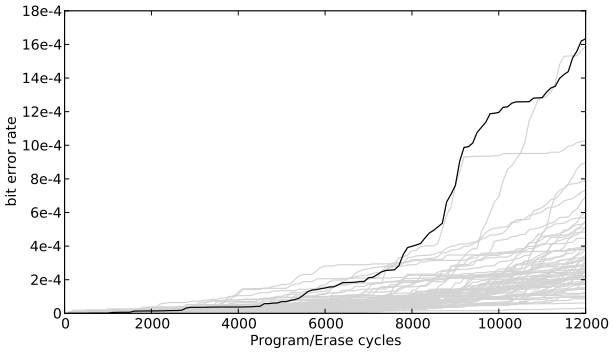

Fig. 6. Bit error rate trend with growing number of extended P/E cycles sustained by the same block, with 8 page reprograms per cycle. Every line represents the BER of a page in the block, with the darker one being the fastest to wear.

According to the findings of Section III and the FTL architecture described in Section IV-B, we split the program phase of each cycle into two sub-phases: before and after block sealing.

- Before block sealing we program the low pages in random order until we have reprogrammed each page 8 times.
- After the block sealing we program only the high pages one time in order.

In this experiment, we chose to evaluate extended P/E cycles with 8 page reprograms. We assume that it is not necessary to reprogram pages beyond this limit because the techniques used to generate WOM compatible data show diminishing returns for higher number of reprograms [3].

After every program operation, the entire block is checked for errors against a golden copy stored on a separate drive. The golden copy is updated, when an error occurs, to avoid counting the same error again in the next iteration.

Ideally, this analysis would be performed with all error protection mechanisms disabled, so that the BER observed corresponds to the real reliability characteristic of the flash chip. Unfortunately, we could not disable the ECC when writing entire pages. To exclude the ECC protection from the BER measurement, at the beginning of the program phase we programmed all low pages with all 1s, and then flipped 50% of the bits with the first reprogram operation. This procedure overloads the error correction capabilities of the ECC used by the OpenSSD, and therefore the following read operation returns the data unmodified and triggers an ECC fail interrupt, which is ignored.

The data reprogrammed is artificially generated to be overwrite compatible with what was programmed before. At every step the previous content of the page is used as a starting point, and the bits with a value of 1 are flipped to 0 with a 50% probability to simulate an update involving half of the bits which are still usable. This is the characteristic of the bit flip pattern expected when using WOM codes.

Figure 6 shows the result of the experiment. Every line in the graph represents the BER for a different low page. Consistently with the findings of Jimenez et al. [17], we found different BER characteristics for different pages. In the figure, we highlighted the page which wears-out fastest. These pages show a sharp increase of the BER after 9000 extended P/E cycles, growing rapidly to 10^{-3}. However, at that point the bulk of the pages show BER of 10^{-4} in average. This result is in line with the traditional analysis presented in previous studies, on similar chips. Therefore, we deduce that, using the page program order we suggested and the block seal operation, it is possible to apply the extended P/E cycles to MLC flash without impact on the BER.

As a corollary of this result, we deduce that the erase operations are the major cause of wear-out because increasing the number of program operations per cycle, keeping the same number of erase operations, does not have significant impact on the BER. Unfortunately, we could not compare this result with the traditional BER analysis performed on the same chip because we could not disable the ECC protection on the first program operation.

In 2009, Grupp et al. [6] published an extensive analysis of different chips finding a large variation among different manufacturers. They observed a sharp BER increase after 9,000 P/E cycles, which is in line with our experiment. Before that point all the MLC chips they tested had BER below 10^{-5}, while in our case most pages have BER of 10^{-4}. However, the same group published a more recent study [1] in which they underlined how, as the feature size of the flash cells scales down, performance and reliability gets worse. In particular, for 35 nm technology, corresponding to our flash chips, the average BER they measured is 10^{-4}, perfectly in line with our experiments.

The only other experiment with repeated program operation per erase cycle, to the best of our knowledge, is the one presented by Jagmohan et al. in [2]. They found BERs ranging from 10^{-9} to 10^{-6}, however their results cannot be compared with ours because they were performed on SLC flash, which is more robust than MLC.

B. FTL Evaluation

In this section, we present experiments comparing the two FTLs presented in Section IV: the baseline FTL and the seal FTL.

The goal of our analysis is to evaluate how well the seal FTL can delay sealing the overwrite blocks to exploit the extended P/E cycles. Furthermore, we examine if the presence of the overwrite blocks hinders the performance of the FTL. Clearly, the overwrite command places a high burden on the FTL and should not be used lightly. Allocating an overwrite block, while the FTL is under heavy load, is more challenging than allocating a write block. While the latter can be obtained efficiently by sealing an overwrite block, the first requires a garbage collected block. The results we present here identify the characteristics of the workloads that can overcome these drawbacks and benefit from the extended P/E cycles.

OpenSSD configuration

For the experiments presented in this section we configured the OpenSSD to use 32 banks with 64 blocks of addressable space. Additionally, every bank has 8 blocks of overprovisioning

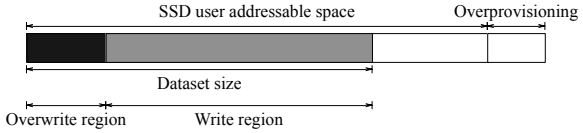

Fig. 7. Parametric workload.

space. These parameters sum up to a total of 8 GB of addressable space with 1 GB of overprovisioning space. The OpenSSD has limited DRAM, enough to map 8 GB of addressable space using the page mapping scheme we implemented. We did not use techniques to compress the mapping table, such as map caching [14] or hybrid mapping [15], because we focus here only on the effects of the extended P/E cycles.

Benchmark description

We used a parametric benchmark that represents a dataset partitioned into an overwrite region and a write region, as depicted in Figure 7. The dataset size for this evaluation has been fixed to 6 GB, that represets 75% of the SSD capacity. The data in the overwrite region is generated artificially to be WOM compatible, and can be written to the flash using the SATA overwrite command, while for the write region the SATA write command must be used. The workload is composed of write operations within the dataset size. Each write operation is aligned on a 32 KB boundary and has size 32 KB, which is the same page size supported by the OpenSSD controller. Thus each operation corresponds to exactly one flash page write, avoiding any misalignment effect. The write operations hit the overwrite or the write regions according to the parameters described below. Once one of the two regions is selected, the address inside it is chosen with uniform distribution across the entire region.

The benchmark characteristics are controlled by two parameters.

- The *overwrite region percentage* parameter controls the size of the overwrite region. It is expressed as a percentage of the dataset size. In this section we present results for overwrite region percentages varying from 5% to 50%.
- The *overwrite skewness* controls the probability that an operation hits the overwrite region. In this section we present results for skewness of 40%, 60%, and 80%.

Every run of the benchmark starts with a completely empty SSD. We warmup the SSD by writing the entire write region one time sequentially, followed by the overwrite region, also written one time sequentially. The overwrite region is written using the SATA overwrite command starting at warmup.

The warmup is followed by the main part of the benchmark where we write 12 GB, i.e. twice the dataset size, with 32 KB operations distributed according to the benchmark parameters as described above. For each set of parameters we generated one trace, and run it against both FTLs.

The benchmark uses artificial data. Every write operation writes random data containing 50% 0s and 50% 1s to the SSD.

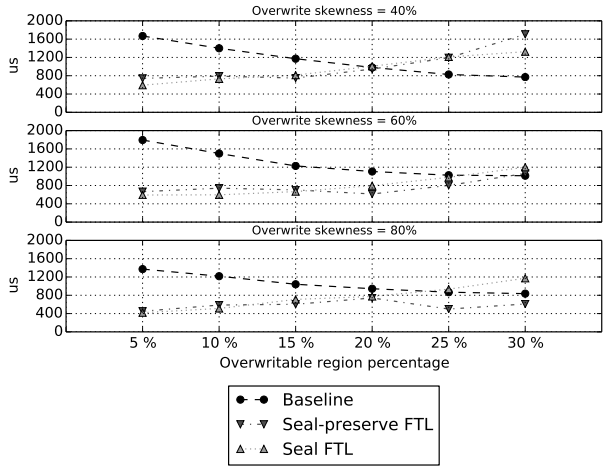

Fig. 8. Latencies in μs for different victim selection policies. *Seal-preserve FTL* refers to the FTL which counts the empty high pages as valid. *Seal FTL* refers to the FTL which considers the empty high pages as invalid.

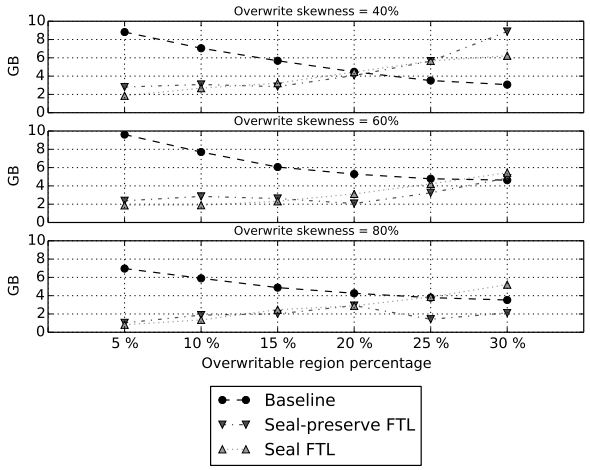

Fig. 9. Amount of data moved during garbage collection operations in GB for different victim selection policies. *Seal-preserve FTL* refers to the FTL which counts the empty high pages as valid. *Seal FTL* refers to the FTL which considers the empty high pages as invalid.

The overwrite operations use this same data pattern for the first time. Later operations on the same address use overwrite compatible data generated by flipping the bits still at 1 into 0 with a 50% probability.

Preserve unused High Pages during GC

The first FTL design decision we evaluate is how to count the unused high pages contained in the full overwrite blocks when selecting a garbage collection victim. As explained in Section IV we examine two approaches: preserving the high pages by counting them as containing valid data, or not preserving them by considering them as invalid. In the first case, the goal is to increase the probability that a write block is chosen as victim, thus putting a premium on the reuse of the overwrite block high pages.

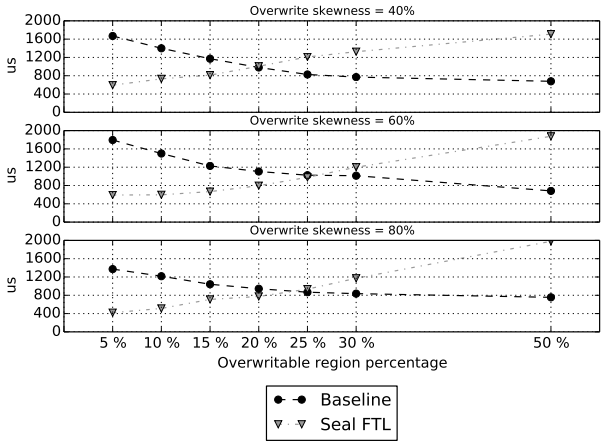

Fig. 10. Average latency (μs) of all the write operations. This includes also the operations executed using the SATA overwrite command in the benchmark run on the extended P/E enabled SSD.

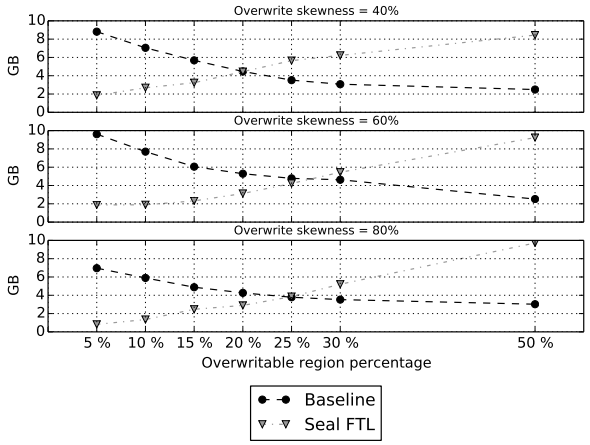

Fig. 11. Amount of data (GB) moved by garbage collection procedures during the entire benchmark run.

The graphs presented in Figure 8 and 9 show, respectively, the average latency per operation and the total amount of data moved by garbage collections in three cases. The lines labeled as *seal-preserve FTL* refer to the FTL counting the empty high pages as containing valid data. The FTL which counts the high pages as invalid is simply labeled as *seal FTL* because it shows the best behavior and is the one considered in the rest of the evaluation.

The first thing to observe is that both FTLs show the same trend: they offer the biggest benefit, with respect to the baseline FTL, when the overwrite region percentage is low, and reduce their performance with a growing overwrite region percentage. The seal FTL is better than the seal-preserve FTL, for all overwrite skewness values, when the overwrite region percentage is smaller than 20%. Above that watermark the seal-preserve FTL has better performance than the seal FTL for high overwrite skewness (significantly better for 80% overwrite skewness). However, when the overwrite percentage grows above 30% the performance of the seal-preserve FTL degrade so much that the OpenSSD becomes unresponsive causing the crash of the benchmark run. This is caused by the fact that when the overwrite region percentage grows, the number of full write blocks decreases, therefore they contain more valid pages. At this point the seal-preserve FTL does not seal the full overwrite blocks because considers all their pages as valid, and therefore triggers garbage collections on full write blocks containing many valid pages. Every garbage collection procedure cleans a few pages, and therefore the FTL has to invoke the garbage collection every few operations, increasing the latency to the point where the device becomes unresponsive.

Our conclusion is that the seal-preserve FTL is very sensitive to the parameters of the workload. It is the better choice for a restricted set of workload characteristics, i.e. between 20% and 30% overwrite region percentage in our parametric workload, and overwrite skewness higher than 60%. It is possible to fine tune the seal FTL to switch to the seal-preserve policy in those conditions, however in the rest of the evaluation we consider only the seal FTL.

Performance

Figures 10 and 11 show the performance of the baseline FTL compared to the seal FTL. The graphs show, respectively, the average write latency and the amount of data moved during the garbage collection operations. The results show that the latency depends directly on the data copied during garbage collection.

The first thing to notice in these graphs is that the trend of both metrics holds for different overwrite skewness values. With higher overwrite skewness the seal FTL exploits more page reprograms on the low pages of the overwrite blocks.

However, the parameter that mostly affects performance is the overwrite space percentage. When this percentage is low, the extended P/E cycles result in a great performance benefit. We measured the highest benefit with an overwrite space percentage of 5% and an overwrite skewness of 60%. In those conditions, the seal FTL exhibits a reduction of 67% in latency and 80% in the amount of data copied during garbage collection. With higher overwrite skewness, the extended P/E cycles technique is even more effective, but the benefit compared to the baseline FTL is not as big.

When the overwrite space percentage increases, the performance of the seal FTL decrease rapidly. To explore the reasons for the performance drop we extracted and examined two additional metrics.

Distance between overwrite operations: Figure 12 shows the distribution of the distances between two overwrite operations on the same location. The distances are measured as GB that are written between the two operations. This metric is very similar to the cache reuse distance used to evaluate caching effectiveness. Ideally, when a page is written in an overwrite block, the FTL should be able to allocate space equal to this distance in other blocks, before sealing it. This way the page can be reprogrammed. An important role in this regard is played by the overprovisioning space and the portion of addressable space not occupied by the dataset. In this evaluation, the overprovisioning space is set to 1 GB,

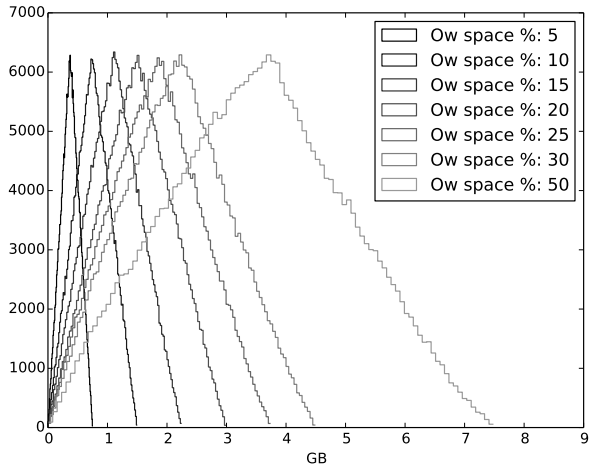

Fig. 12. Distribution of the distance between overwrites on the same location. It is measured as the amount of data (GB) written between the two operations. This graph reports the values for overwrite skewness 80%.

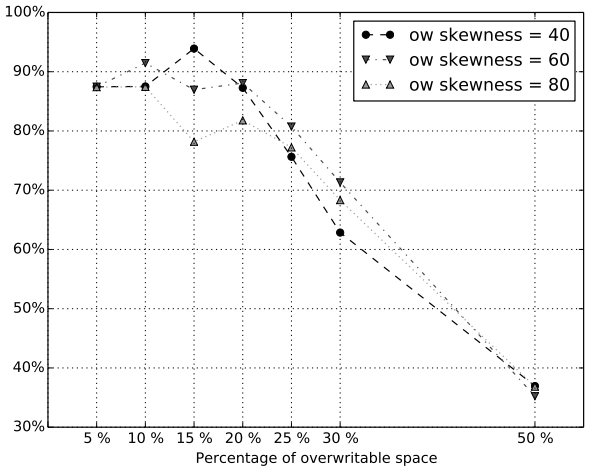

Fig. 13. Percentage of overwrite commands that are translated into flash reprogram operations.

while the unoccupied portion is 2 GB. The graph shows that in the scenario with 5% overwrite space the distance is well below the 3 GB watermark. When the distance approaches or surpasses the watermark, the performance drops. This metric is extremely important for applications willing to exploit the extended P/E cycles. The access pattern must be analyzed to ensure that the distance between overwrite operations presents a favorable profile.

Page reprogram percentage: The second metric, plotted in Figure 13, is the percentage of overwrite operations that are successfully translated by the FTL into page reprograms. Recall that not all the overwrite operations result in page reprograms. For instance, when the overwrite blocks are sealed, the pages contained therein can not be reprogrammed. Another cause of missed reprogram is that after 8 consecutive reprogram operations to the same page, the page is reallocated to a different overwrite block, to simulate the ECC limitations,

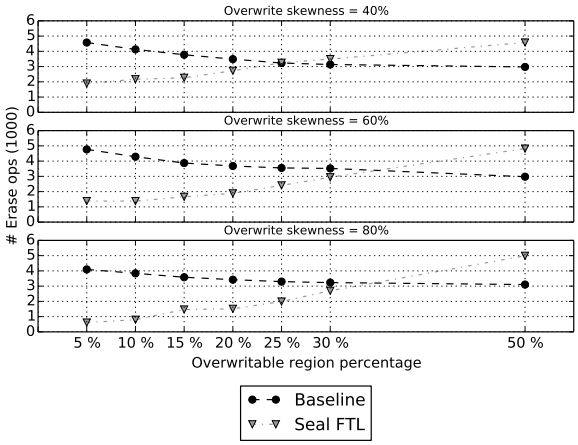

Fig. 14. Number of garbage collection operations, equivalent to the number of erase operations.

as described in Section IV. This graph, according to the other metrics, shows that when the overwrite space percentage increases it is more difficult for the FTL to reprogram pages. The overwrite operations are spread across a higher portion of the data set, and require more overwrite blocks. In turn, the accumulation of overwrite blocks reduces the number of the write blocks that will contain, in average, more valid data. As a consequence of this increase, more overwrite blocks are prematurely chosen as garbage collection victims, because the overwrite blocks can have at maximum half the valid data contained in the write blocks (only half pages are used before sealing). When the write blocks have more than half valid pages, the overwrite blocks are necessarily chosen as victims.

Endurance

Figure 14 shows the number of garbage collection operations. The results show the same trend observed in the performance evaluation, therefore the same considerations discussed before apply here. Every garbage collection invocation corresponds to one block erase operation. The FTL does not perform erase operations in any other circumstance. Therefore, a reduction in garbage collections causes a direct increase in endurance, since the erase operations are the major cause of wearout (as deduced from our BER analysis). We observed a garbage collection reduction of 71% and 85% with overwrite skewness of 60% and 80%, respectively, when the overwrite space percentage is 5%.

It is interesting to note that the stark reduction of the total amount of data copied by garbage collection shown in Figure 11 is caused only by the reduction of the number of garbage collection operations, and not by a reduction of data moved by each of them. Figure 15 shows the amount of data copied in average by single garbage collection operations. The baseline FTL shows a decreasing trend with a growing overwrite region percentage, while the seal FTL shows an increasing trend and exceeds the baseline FTL starting with overwrite region percentage of 10% for overwrite skewness of 80%. However, with the same overwrite skewness, the total amount of data

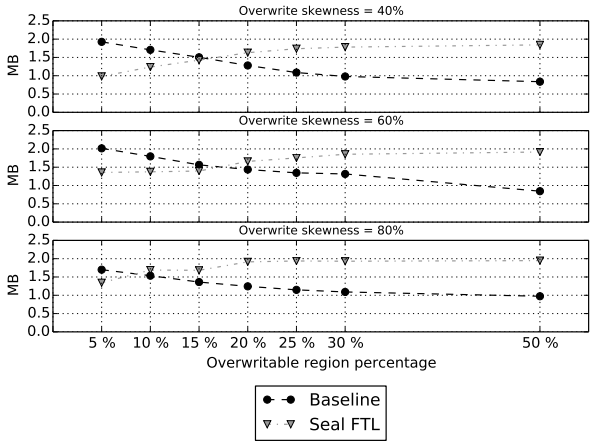

Fig. 15. Average amount of data (MB) moved per garbage collection.

copied during garbage collection by the seal FTL meets and exceeds the baseline FTL only when the overwrite region percentage is greater than 25%. Between these two overwrite region percentages the benefit of the seal FTL is obtained only from the sheer reduction of garbage collection invocations. Similar considerations apply to the other overwrite skewness values.

Read performance

The read performance is not affected by the extended P/E cycles, as they are implemented in this work, because the size of the data is not modified by the FTL and there is no additional step involved (e.g. no WOM encoding is performed within the FTL). Thus, the analysis presented here does not consider read operations. However, if the application requires some space overhead or additional steps to generate WOM compatible data then the read and write performance may be affected. The evaluation of such effects is out of the scope of this work.

VI. Related Work

FTL development received considerable attention in academic literature after the appearance of the first commercial SSDs. Our technique is orthogonal to the mapping technique and therefore can be applied to most schemes [14], [15], [18], [19]. In our implementation we opted for a plain page-mapped scheme to avoid the effect of a particular FTL optimization on the results.

The FTL we presented is designed to accept overwrite commands through the SATA interface. Therefore, it delegates the responsibility for generating WOM compatible data to the application. Other approaches have been proposed: Odeh at al. and Yadgar et al. presented designs where the FTL internally uses WOM codes to perform reprogram operations, referred to as *second writes* [4], [7].

Applications using WOM codes [2], [5] can directly use our FTL implementation. However, as shown by our experiments, the highest benefit is obtained when overwrites are concentrated in a small portion of the dataset. Therefore, it is advisable to use WOM codes on a restricted and frequently updated subsection of it, as in [4], [7].

The reprogram operation was discussed by Jagmohan et al., Kaiser et al., Odeh et al, and Yadgar et al. in [2]–[4], [7]. However, those studies are based on simulations and do not attempt to implement the operation on a real prototype. To the best of our knowledge this work is the first attempt at doing so. Furthermore Jagmohan et al. discuss the reprogram operation, referred to as *eraseless program*, only in SLC flash chips.

The diagram presented in Figure 3 is consistent with the findings of Grupp et al. presented in [6]. Their experiments consisted of erasing a block and repeatedly programming half of one page to 0. They present two results, which can be explained by the transitions outlined in our diagram:

- reprogramming the low pages does not disturb other pages. This case corresponds to one first use of transition 1, and then repeated uses of transition 3 with the intent to program the low bit to 0. Both transitions are safe, and therefore no error was detected.
- reprogramming the high page disturbs the bits of the corresponding low page after two reprograms. This case corresponds to one first use of transition 2, that is safe, and then one use of transition 7 which is unsafe because it causes program disturb on the low bit. Therefore, the errors detected are in line with our state diagram.

Kaiser et al. present a B-Tree that uses unsorted nodes and append new key-value pairs in the node. This approach allows to write the nodes with the SATA overwrite command, because appending is by definition WOM compatible. This B-Tree could directly be stored on a SSD with the seal FTL. In the original publication the authors considered a simplified model where every flash page could be reprogrammed.

VII. Conclusion

In this work we presented the extended P/E cycles technique implemented through page reprograms. This technique can cause a loss of capacity in MLC flash since only the low pages of a block can be used for reprograms. We showed how to circumvent this drawback through overwrite block sealing. This technique enables the utilization of the complete capacity of blocks whose pages were previously reprogrammed.

During the development of the overwrite block sealing, we extracted the state transition diagram for MLC flash. So far, not all the transitions in the diagram are used. However, the diagram can be the foundation for data encoding research aimed at utilizing all the transitions and increase the MLC utilization.

The BER analysis we performed showed that the erase voltage is the major contributor to flash cell wear. Extended P/E cycles have the same number of erase operations as normal P/E cycles, but can write more data per cycle, therefore increasing the lifetime of flash cells. Our evaluation shows that this technique can reduce erase operations by as much as 85% on workloads with certain characteristics. Furthermore, when used in optimal conditions, it can lead to latency reduction as

high as 67% with up to 60% less data copied internally by the FTL.

The technique presented here is not directly applicable to every application. First, the data must be WOM compatible, and this requires some special data structure design, or encoding part of the dataset with WOM codes. Second, the access pattern must exhibit some specific properties, detailed in Section V. However, we believe that the benefits are such that designers seeking to squeeze the best performance and lifetime from flash devices should consider this technique and the insights presented here.

The source code for the FTLs presented in this paper is available at https://github.com/zdvresearch.

VIII. Acknowledgement

We would like to thank our shepherd Gala Yadgar for her guidance and support, and the anonymous reviewers for their insightful comments.

References

[1] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, p. 2.

[2] A. Jagmohan, M. Franceschini, and L. Lastras, "Write amplification reduction in NAND flash through multi-write coding," in *Proceedings of the 26th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–6.

[3] J. Kaiser, F. Margaglia, and A. Brinkmann, "Extending SSD lifetime in database applications with page overwrites," in *Proceedings of the 6th Annual International Systems and Storage Conference (SYSTOR)*, 2013, p. 11.

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

[5] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Information and Control*, vol. 55, no. 1-3, pp. 1–19, 1982.

[6] 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 *Proceedings of the 42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 24–33.

[7] S. Odeh and Y. Cassuto, "NAND flash architectures reducing write amplification through multi-write codes," in *Proceedings of the 30th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2014.

[8] The OpenSSD Project, "Indilinx Jasmine platform," *www.openssd-project.org*.

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

[10] J. Hsieh, T. Kuo, and L. Chang, "Efficient identification of hot data for flash memory storage systems," *ACM Transactions on Storage (TOS)*, vol. 2, no. 1, pp. 22–40, 2006.

[11] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] E. Yaakobi, J. Ma, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf, "Error characterization and coding schemes for flash memories." in *GLOBECOM Workshops (GC Wkshps), IEEE*, 2010, pp. 1859–1860.

[13] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, Jul. 2007.

[14] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2009, Washington, DC, USA, March 7-11, 2009*, pp. 229–240.

[15] S.-P. Lim, S.-W. Lee, and B. Moon, "FASTer FTL for enterprise-class flash memory SSDs," in *Proceedings of the 6th IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2010, pp. 3–12.

[16] E. E. Gad, Y. Li, J. Kliewer, M. Langberg, A. Jiang, and J. Bruck, "Polar coding for noisy write-once memories," *Proceedings of IEEE International Symposium on Information Theory (ISIT), 2014.*

[17] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: improving NAND flash lifetime by balancing page endurance," in *Proceedings of the 12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014*, 2014, pp. 47–59.

[18] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 3, 2007.

[19] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives." *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, vol. 11, 2011.