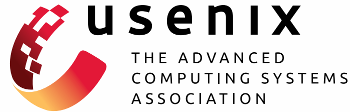

Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance

Xavier Jimenez, David Novo, and Paolo Ienne,
Ecole Polytechnique Fédérale de Lausanne (EPFL)

https://www.usenix.org/conference/fast14/technical-sessions/presentation/jimenez

This paper is included in the Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14).

February 17–20, 2014 • Santa Clara, CA USA

ISBN 978-1-931971-08-9

Open access to the Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14) is sponsored by

Wear Unleveling:
Improving NAND Flash Lifetime by Balancing Page Endurance

Xavier Jimenez, David Novo and Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH–1015 Lausanne, Switzerland

Abstract

Flash memory cells typically undergo a few thousand *Program/Erase* (P/E) cycles before they wear out. However, the programming strategy of flash devices and process variations cause some flash cells to wear out significantly faster than others. This paper studies this variability on two commercial devices, acknowledges its unavoidability, figures out how to identify the weakest cells, and introduces a wear unbalancing technique that let the strongest cells relieve the weak ones in order to lengthen the overall lifetime of the device. Our technique periodically skips or relieves the weakest pages whenever a flash block is programmed. Relieving the weakest pages can lead to a lifetime extension of up to 60% for a negligible memory and storage overhead, while minimally affecting (sometimes improving) the write performance. Future technology nodes will bring larger variance to page endurance, increasing the need for techniques similar to the one proposed in this work.

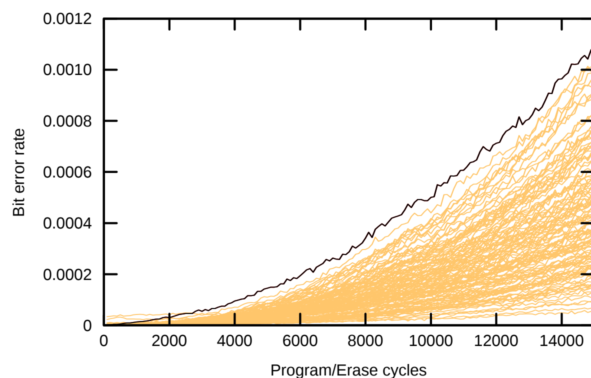

Figure 1: **Page degradation speed variation.** These data were generated by continuously writing random values into the 128 pages of a single block of flash. The BER grows at widely different speeds among pages of the same block. We suggest to reduce the stress on the weakest pages in order to enhance the block endurance.

1 Introduction

NAND flash is extensively used for general storage and transfer of data in memory cards, USB flash drives, solid-state drives, and mobile devices, such as MP3 players, smartphones, tablets or netbooks. It features low power consumption, high responsiveness and high storage density. However, flash technology also has several disadvantages. For instance, devices are physically organized in a very specific manner, in blocks of pages of bits, which results in a coarse granularity of data accesses. The memory blocks must be erased before they are able to program (i.e., write) their pages again, which results in cumbersome out-of-place updates. More importantly, flash memory cells can only experience a limited number of *Program/Erase* (P/E) cycles before they wear out. The severity of these limitations is somehow mitigated by a software abstraction layer, called a *Flash Translation Layer* (FTL), which interfaces between common file systems and the flash device.

This paper proposes a technique to extend flash devices' lifetime that can be adopted by any FTL mapping the data at the page level. It is also suitable for hybrid mappings [13, 6, 12, 5], which combine page level mapping with other coarser granularities.

The starting point of our idea is the observation that the various pages that constitute a block deteriorate at significantly different speeds (see Figure 1). Consequently, we detect the weakest pages (i.e., the pages degrading faster) to relieve them and improve the yield of the block. In essence, to relieve a page means not programming it during a P/E cycle. The idea has a similar goal as wear leveling, which balances the wear of every block. However, rather than balancing the wear, our technique carefully unbalances it in order to transfer the stress from weaker pages to stronger ones. This means

that every block of the device will be able to provide its full capacity for a longer time.

The result is a device lifetime extension of up to 60% for the experimented flash chips, at the expense of negligible storage and memory overheads, and with a stable performance. Importantly, the increase of process variations of future technology nodes and the trend of including a growing number of pages in a single block let us envision an even more significant lifetime extension in future flash memories.

2 Related Work

Flash lifetime is one of the main concerns of these devices and is becoming even more worrisome today due to the increasing variability and retention capability inherent to smaller technology nodes. Most of the techniques trying to improve the device lifetime focus on improving the ECC robustness [15, 26], on reducing garbage collection overheads [14, 25], or on improving traditional wear-leveling techniques [20]. All of these contributions are complementary to our technique.

Lue et al. suggest to add a built-in local heater on the flash circuitry [16], which would heat cells at 800 ˚C for milliseconds to accelerate the healing of the accumulated damage on the oxide layer that isolates the floating gates. Based on prototyping and simulations, the authors envision a flash cell endurance increase of several orders magnitude. While the endurance improvement is impressive, it would require significant efforts and modifications in current flash architectures before being available on the market. Furthermore, further analysis (e.g., power, temperature dissipation, cost) might reveal constraints that are only affordable for a niche market, whereas our technique can be used today with off-the-shelf NAND flash chips.

Wang and Wong [24] combine the healthy pages of multiple bad blocks to form a smaller set of virtually healthy blocks. In the same spirit, we revive *Multi-Level Cell* (MLC) bad blocks in *Single-Level Cell* (SLC) mode in a previous work [11]: writing a single bit per cell is more robust and can sustain more stress before a cell becomes completely unusable. Both techniques wait for blocks to turn bad before acting, which somehow limits their potentials (17% lifetime extension at best); on the other hand, by relieving early the weakest pages, we benefit more from the strongest cells and thus show a better lifetime improvement.

Pan et al. acknowledge the block endurance variance and suggest to adapt classical wear-leveling algorithm to compare blocks on their *Bit Error Rate* (BER) rather than their P/E cycles count [20]. However, in order to monitor a block BER, the authors assume homogeneous page endurance and a negligible faulty bit count variance between P/E cycles. For the two chips we studied, both assumptions were not applicable and would require a more complex approach to compare the BER of multiple blocks. Furthermore, we observed a significantly larger endurance variance on the page level than the block level. Hence, by acting on the page endurance, our approach has more room to expand the device lifetime.

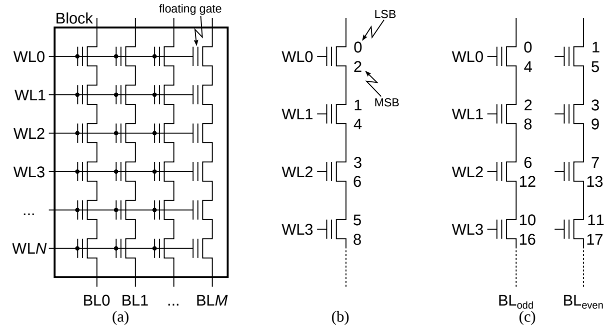

Figure 2: **Flash cells organization.** Figure 2(a) shows the organization of cells inside a block. A block is made of cell strings for each bitline (BL). Each bit of an MLC is mapped to a different page. Figures 2(b) and 2(c) show two examples of cell-to-page mappings in 2-bit MLC flash memories. For instance, in Figure 2(b), the LSB and MSB of WL_1 are mapped to pages 1 and 4, respectively. The page numbering also gives the programming order.

In this work, for more efficiency, we restrict the relief mechanism to data that is frequently updated, which is a strategy shared with techniques proposing to allocating those data in SLC-mode (i.e., programming only one bit per cell) to reduce the write latency [9, 10]. In a previous work, we characterized the effect of the SLC-mode and observed that it could write more data for the same amount of wear compared to regular writes and provided a lifetime improvement of up to 10% [10]. In this work, we propose to go further in the lifetime extension.

3 NAND Flash

NAND flash memory cells are grouped into pages (typically 8–32 kB) and blocks of hundreds of pages. Figure 2(a) illustrates the cell organization of a NAND flash block. In current flash architectures, more than one page can share the same *WordLine* (WL). This is particularly true for *Multi-Level Cells* (MLC), where the *Least Significant Bits* and *Most Significant Bits* (LSB and MSB) of a cell are mapped to different pages. Figures 2(b) and 2(c) show two cell-to-page mappings used in MLC flash devices, *All-BitLine* (ABL) and *interleaved*, respectively.

Flash memories store information by using electron tunneling to place and remove charges into floating gates.

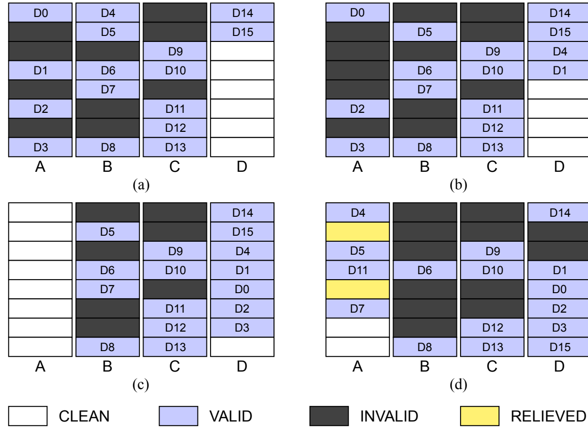

Figure 3: **Pages state transitions.** Figure (a) shows the various page states found in typical flash storage: *clean* when it has been freshly erased, *valid* when it holds valid data, and *invalid* when its data has been updated elsewhere. In Figure (b), data D1 and D4 are invalidated from blocks A and B, and updated in block D. In Figure (c), block A is reclaimed by the garbage collector; its remaining valid data are first copied to block D, before block A gets erased. Figure (d) illustrates the mechanism proposed in this work: we opportunistically relieve weak pages to limit their cumulative stress.

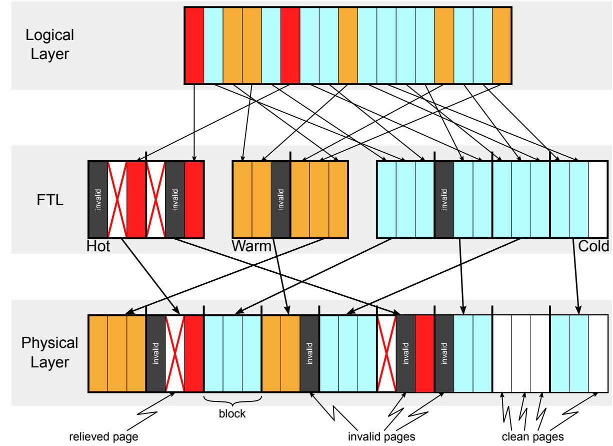

Figure 4: **Flash Translation Layer example.** An example of page-level mapping distinguishing update frequencies in three categories: *hot*, *warm* and *cold*. In this work, we propose to idle the weakest pages when their corresponding block is allocated to the hot partition. It limits the capacity loss to a small portion of the storage but still benefits from high update frequency to increase page-relief opportunities.

The action of adding a charge to a cell is called *programming*, whereas its removal is called *erasing*. Reading and programming cells is performed on the page level, whereas erasing must be performed on an entire block. Furthermore, pages in a block must be programmed sequentially. The sequence is designed to minimize the programming disturbance on neighboring pages, which receive undesired voltage shifts despite not being selected. In the sequences defined by both cell-to-page mappings, the LSBs of WL_{i+1} are programmed before the MSBs of WL_i. In this manner, any interference occurring between the WL_i LSB and MSB program will be inhibited after the WL_i MSB is programmed [17].

Importantly, the flash cells have limited endurance: they deteriorate with P/E cycles and become unreliable after a certain number of such cycles. Interestingly, the different pages of a block deteriorate at different rates, as shown in Figure 1. This observation serves as motivation for this work, which proposes a technique to reduce the endurance difference by regularly relieving the weakest pages.

3.1 Logical to Physical Translation

Flash Translation Layers (FTLs) hide the flash physical aspects to the host system and map logical addresses to physical flash locations to provide a simple interface similar to classical magnetic disks. To do this, the FTL needs to maintain the state of every page—typical states are *clean*, *valid*, or *invalid*, as illustrated in Figure 3(a). Only clean pages (i.e., erased) can be programmed. Invalid and valid pages cannot be reprogrammed without being erased before, which means the FTL must always have clean pages available and will direct incoming writes to them. Whenever data is written, the selected clean page becomes valid and the old copy becomes invalid. This is illustrated in Figure 3(b), where D1 and D4 have been reallocated.

To enable our technique, we introduced a fourth page state, *relieved*, to indicate pages to be relieved (i.e., not programmed) during a P/E cycle. Relieving pages during a P/E cycle is perfectly practical, because it does not break the programming sequentiality constraint and does not compromise the neighbors information. In fact, it is electrically equivalent to programming a page to the erase state (i.e., all 1's). Hence, to the best of our knowledge, any standard NAND flash architecture should support this technique.

3.2 Garbage Collection

The number of invalid pages grows as the device is written. At some point, the FTL must trigger the reuse of invalid pages into clean pages. This reuse process is known as garbage collection, which is illustrated in Figure 3(c), where block A is selected as the victim.

Copying the remaining valid data of a victim block represents a significant overhead, both in terms of performance and lifetime. Therefore, it is crucial to select the data that will be allocated onto the same block carefully in order provide an efficient storage system. Wu and Zwaenepoel addressed this problem by regrouping data with similar update frequencies [25]. *Hot* data have a higher probability of being updated and invalidated soon, resulting in *hot* blocks with a large number of invalid pages that reduce the garbage collection overhead. Figure 4 shows an example FTL that identifies three different *temperatures* (i.e., update frequencies), labeled as *hot*, *warm*, and *cold*. Literature is rich with heuristics to identify hot data [12, 4, 9, 22, 21].

In the present study, we propose to relieve the weakest pages in order to balance their endurance with their stronger neighbors. We have restricted the relieved pages to the hottest partition in order to limit the resulting capacity loss to a small and contained part of the storage, while benefiting from a large update frequency to better exploit the presented effect. Following sections will further analyze the costs and benefits of our approach, as well as its challenges.

3.3 Block Endurance

While accumulating P/E cycles, a block becomes progressively less efficient in the retention of charges and its BER increases exponentially. Typically, flash blocks are considered unreliable after a specified number of P/E cycles known as the endurance. Yet, it is well understood that the endurance specified by manufacturers serves as a certification but is hardly sufficient to evaluate the actual endurance of a block [8, 18]. A block endurance depends on the following factors: First, the cell design and technology will define its resistance to stress; this is generally a trade-off with performance and density. Second, the endurance is associated with a retention time, that is, how long data is guaranteed to remain readable after being written; a longer retention time requirement will require relatively healthy cells and limit the endurance to lower values. Finally, ECCs are typically used to correct a limited number of errors within a page; the ECC strength (i.e., number of correctable bits) influences the block endurance. The ECC strength required to maintain the endurance specified by manufacturers increases drastically at every new technology nodes. A stronger ECC grows in size and requires a more complex and longer error decoding process, which compromises read latency. Additionally, the strength of an ECC is chosen according to the weakest page of a block and, as suggested by Figure 1, the chosen strength will only be justified for a minority of pages. Our proposed balancing of page endurance within a block will reduce the BER of the weakest pages; therefore, our idea can either be used to reduce the ECC strength requirement or to extend the device lifetime. However, in this work, we only explore the impact of our technique in device lifetime extension.

FTLs implement several techniques that maximize the use of this limited endurance to guarantee a sufficient device lifetime and reliability. Typical wear-leveling algorithms implemented in FTLs target the even distribution of P/E counts over the blocks. Additionally, to avoid latent errors, *scrubbing* [1, 23] may be used, which consists in detecting data that accumulates too many errors and rewriting it before it exceeds the ECC capability.

3.4 Bad Blocks

A block is considered *bad* whenever an erase or program operation fails, or when the BER grows close to the ECC capabilities. In the former case, an operation failure is notified by a status register to the FTL, which reacts by marking the failing block as bad. In the latter case, despite a programming operation having been completed successfully, a certain number of page cells might have become too sensitive to neighboring programming disturbances or have started to leak charges faster than the specified retention time and will compromise the stored data [17]. Henceforth, the FTL will stop using the block and the flash device will die at the point in time when no spare blocks remain to replace all failing blocks.

To study the degradation speed of the different pages within a block, we conducted an experiment on a real NAND flash chip in which we continuously programmed pages with random data and monitored each page BER by averaging their error counts over 100 P/E cycles. We have already anticipated the results in Figure 1, which shows how the number of error bits increases with the number of P/E operations for all the pages in a particular block. At some point in time, the weakest page (darker line on the graph) will show a BER that is too high and the entire block will be considered unreliable. Interestingly, a large majority of the remaining pages could withstand a significant amount of extra writes before becoming truly unreliable. Clearly, flash blocks suffer a premature death if no countermeasures are taken and our approach attempts to postpone the moment at which a page block becomes bad by proactively relieving its weakest pages. The following sections further study the degradation process of individual pages and detail the technique that uses strong pages to relieve weak ones.

4 Relieving Pages

In this section we introduce the relief strategy and characterize its effects from experiments on two real 30-nm class NAND flash chips.

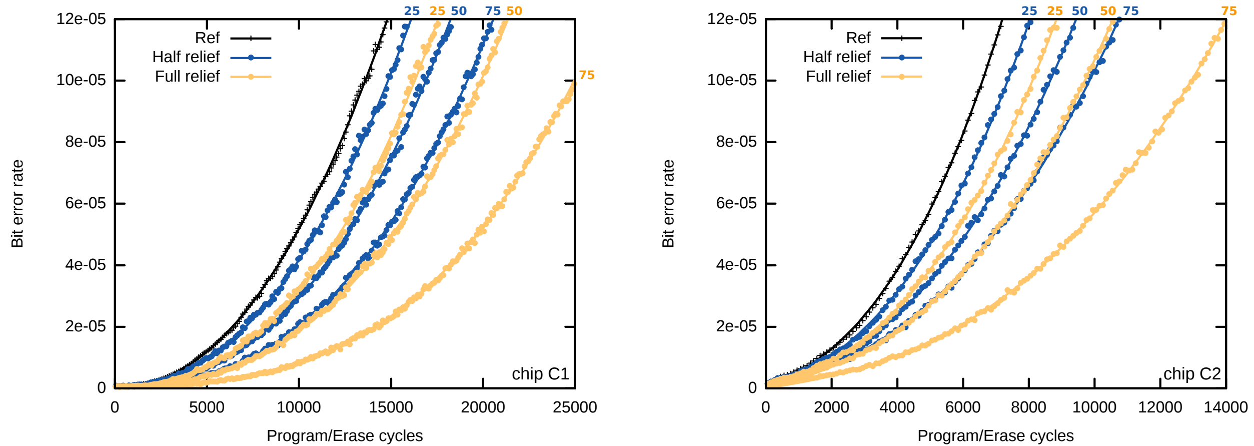

Figure 5: **Measured effect of relieving pages.** The degradation speed for various relief rates and types are measured on both chips. The *Ref* curve reports the BER of the entire reference blocks, whereas for the relieved blocks, the BER is only evaluated on the relieved page. The labels '25', '50', and '75' indicate the corresponding relief rate in percent. The BER is evaluated over a 100-cycle period.

4.1 Definition

We define a *relief* cycle on a page the fact of not programming it between two erase cycles. Although relieved pages are not programmed, they are still erased, which, in addition to the disturbances coming from neighbors undergoing normal P/E cycles, generates some stress that we characterize in Section 4.2. In the case of MLC, the cells are mapped to an LSB and MSB page pair and can either be *fully* relieved, when both pages are skipped, or *half* relieved, when only the MSB page is skipped. The level of damage done to a cell during a P/E cycle is correlated to the amount of charge injected for programming; of course, more charges means more damage to the cell. Therefore, a page will experience minimal damage during a *full* relief cycle while a *half* relief cycle will apply a stress level somewhere between the *full* relief and a normal P/E cycle.

4.2 Understanding the Relieving Effect

In order to characterize the effects of relieving pages, we selected two typical 32 Gb MLC chips from two different manufacturers. We will refer them as C1 and C2; their characteristics are summarized in Table 1. The read latency, the block size, and the cell-to-page mapping architecture are the most relevant differences between the two chips. The C1 chip has slower reads and smaller blocks than C2, and it implements the *All-Bit Line* (ABL) architecture illustrated in Figure 2(b). The C2 chip implements the *interleaved* architecture illustrated in Figure 2(c). We design an experiment to measure on our flash chips how the relief rate impacts the page degradation speed. Accordingly, we selected a set of 28 blocks and divided them into seven sets of four blocks each. One set is configured as a reference, where blocks are always programmed normally—i.e., no page is ever relieved. We allocate then three sets for each of the two relief types (i.e., *full* and *half*), and each of these three sets is relieved at a different frequency (25%, 50% and 75%). For each relieved block, only one LSB/MSB page pair out of four is actually relieved, while the others are always programmed normally. Therefore, the relieved page pairs are isolated from each other by three normally-programmed page pairs. Hence, we take into account the impact of normal neighboring pages activity on the relieved pages. Furthermore, within each four-block relieved sets, we alternate the set of page pairs that are actually relieved in order to evaluate evenly the relief effects for every page pair physical position and discard any measurement bias. Finally, every ten P/E cycles we enforce a regular program cycle for every relieved blocks (including relieved pages) in order to average out the absence of disturbance coming from relieved neighbors and collect unbiased error counts for every page. Indeed,

Table 1: MLC NAND Flash Chips Characteristics

Features	C1	C2
Total size	32 Gb	32 Gb
Pages per block	128	256
Page size	8 kB	8 kB
Spare bytes	448	448
Read latency	150 μs	40-60 μs
LSB write lat.	450 μs	450 μs
MSB write lat.	1,800 μs	1,500 μs
Erase latency	4 ms	3 ms
Architecture	ABL	interleaved

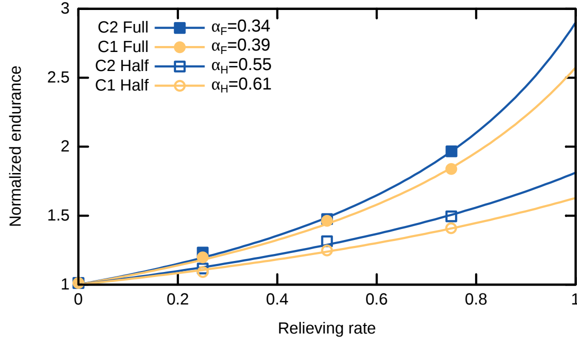

Figure 6: **Normalized page endurance vs. relief rate.** The graph shows how relieving pages extends their endurance. The endurance is normalized to the normal page endurance, corresponding to a maximum BER of 10^{-4}. For each chip, the relative stress of the full and half relief type is extracted by fitting the measured points.

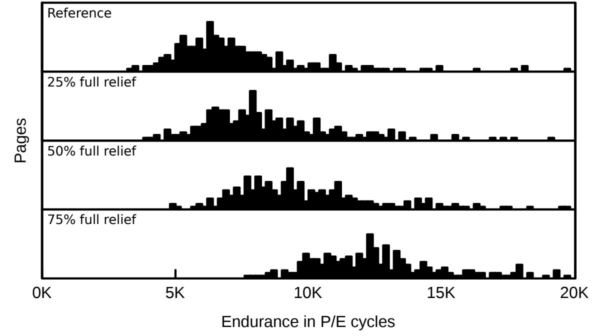

Figure 7: **Measured page endurance distribution.** The clusters on the left and right correspond to MSB and LSB pages, respectively. Both clusters endurance are extended homogeneously when relieved.

pages close to relieved pages experience less disturbance and show a significantly lower BER.

Figure 5 shows the evolution of the average BER with the number of P/E cycles for every set of blocks as measured on the chips. For the relieved sets, only the relieved pages are considered for the average BER evaluation. Clearly, the relief of pages slows down the degradation compared to regular cycles and extends the number of possible P/E cycles before reaching a given BER.

In order to model the stress endured by pages undergoing a full or half relief cycle, we first define the relationship between page endurance and the stress experienced during a P/E cycle. The endurance E of a page is inversely proportional to the stress ω that the page receives during a P/E cycle:

$$E = \frac{1}{\omega}. \tag{1}$$

Considering a page being relieved with a relative stress α at a given rate ρ, the resulting extended endurance E_X is expressed as the inverse of the average stress:

$$E_X(\rho, \alpha) = \frac{1}{(1-\rho)\omega + \rho\alpha\omega} = \frac{E}{(1-\rho) + \rho\alpha}. \tag{2}$$

Assuming a maximum BER of 10^{-4} to define a page endurance, we show in Figure 6 the endurance of relieved pages for the three relief rates measured, with the endurance normalized to the reference set. For each chip, we also fit the data points to the model of Equation (2) and report the extracted α parameters on the figure. Consistently across the two chips, a full relief incurs less damage to the cell than a half relief, which in turn incurs less damage than regular P/E cycles. Interestingly, half reliefs are more efficient than full reliefs in term of stress per written data: for example, for chip C1, the fraction of stress associated to half and full relief cycles is $\alpha_H = 0.61$ and $\alpha_F = 0.39$, respectively. Over two P/E cycles, if an LSB/MSB page pair gets twice half relieved or once fully relieved, two pages would have been written in both cases but the cumulated stress would be larger with a full relief:

$$2 \cdot \alpha_H = 1.22 < 1.39 = 1 + \alpha_F. \tag{3}$$

Furthermore, a half relief cycle consists in programming solely the LSB of a LSB/MSB pair, and, intrinsically, programming the LSB has a significantly smaller latency than the MSB (see Table 1). Thus, a half relief is not only more efficient for the same amount of written data, but it also displays better performance.

Figure 7 provides further insight on the relief effect on a page population. The figure shows the number of P/E cycles tolerated by the different pages before reaching an BER of 10^{-4} evaluated over 100 P/E cycles.

In the next sections we will discuss how relief cycles can opportunistically be implemented into common FTLs to balance the page endurance and improve the device lifetime.

5 Implementation in FTLs

In this section, we describe the implementation details required to upgrade existing FTL with our technique.

5.1 Mitigating the Capacity Loss

Relieving pages during a P/E cycle temporarily reduces the effective capacity of a block. Therefore, relieving pages in a block-level mapped storage would be impractical. Conversely, performing it on blocks that are mapped to the page level (or finer level) is straightforward. Consequently, in order to limit the total capacity loss while still being able to frequently relieve pages,

we propose to exclusively enable relief cycles in blocks that are allocated to the hottest partition, where the FTL writes data identified as very likely to be updated soon.

Actually, the hot partition is an ideal candidate for our technique because of two reasons: (1) hot data generally represent a small portion of the total device capacity (e.g., less than 10%), which bounds the capacity loss to a small fraction; also, (2) hot partitions usually receive a significant fraction of the total writes (our evaluated workloads show often more than 50% of writes identified as hot), which provides plenty of opportunities to relieve pages. Note that flash blocks are dynamically mapped to the logical partitions, and thus, all of the physical blocks in the device will eventually be allocated to the hottest partition. Furthermore, classical wear-leveling mechanisms will regularly swap cold blocks with hot blocks in order to balance their P/E counts. Accordingly, our technique has a global effect on the flash device despite acting only on a small logical partition.

We will now describe two different approaches to balance the page endurance with our relief strategies. The first one can be qualified as *reactive*, in that it will regularly monitor the faulty bit count to identify weak pages. The second one, which we call *proactive*, estimates beforehand what the endurance of every page will be and sets up a relief plan that can be followed from the first P/E cycle. Currently, manufacturers do not provide all the information that would be required to directly specify the parameters needed for our techniques. Until then, both techniques would require some characterization of the chips to be used in order to extract parameters α_F and α_H, and the page endurance distribution.

5.2 Identifying Weak Pages on the Fly

The *reactive* relief technique relies on the evolution of the page BER to detect weakest pages as early as possible. The FTL must therefore periodically monitor the amount of faulty bits per page which is very similar to the scrubbing process [1]. This monitoring happens every time that a cold (i.e., non-*hot*) block is selected by the garbage collector. Concretely, we must read every page and collect the error counts reported by the ECC unit before erasing a block.

A simple approach to identify the weakest pages is to detect which ones reach a particular error threshold first. Assuming that an ECC can handle up to n faulty bits per page, we can set an intermediate threshold k, with $k < n$, that can be used to flag pages getting close to their endurance limit. The parameter n is given by the strength of the ECC in place, while the parameter k must be chosen to maximize the efficiency of the technique and will depend on the page endurance variance. As soon as a page reaches the threshold k, our heuristic will systematically relieve the corresponding LSB/MSB page pair when it is allocated to the hot partition. In order to control the capacity loss, we also set a maximum amount of pages to relieve per block; only the r first pages reaching the threshold within a block will get relieved. For our evaluation, we bound the relieved page count, r, to 25% of the block capacity. A larger r would increase the range of pages that can be relieved but decrease the efficiency of the buffer. Besides, the latest pages to be identified as weak do not require a relief as aggressive than the weakest ones. Hence, we propose to fully relieve the r_h first weak pages and to half relieve the remaining $r - r_h$ pages. In our case, we found the best compromise with r_h equal to 5% and 10% of the block capacity for C1 and C2, respectively. Choosing efficiently r_h for a new chip requires the information on its page endurance distribution. The larger is its variance, the larger r_h should be.

The *reactive* approach requires extra storage for its metadata. This overhead includes two bits per LSB/MSB page pair, which will indicate whether any of the pages has reached the k threshold and whether it should be fully or half relieved, and a (redundant) counter indicating the number of detected weak LSB/MSB page pairs so far. Accordingly, 133 extra bits (128 bits for the flags and 5 bits for the counter) per block will need to be stored in a device containing 128-page blocks. In the concrete case of C1, for instance, this extra storage corresponds to an insignificant amount of the total 458,752 spare bits that are available for extra storage in every block. Additionally, the FTL main memory will need to temporally store the practically insignificant metadata of a single block to be able to restore the metadata after erasing the block. Overall, the extra storage needed by this technique appears to be negligible in typical flash devices.

The monitoring required by this technique needs the FTL to read a whole block before erasing it, which adds an overhead to the erasing time. The monitoring represents an overhead of 10% of the total time spent writing cold data, since flash read latency is typically ten times smaller than write latency. However, the monitoring process can often be performed in the background, making this estimation—which we will use in all of our experiments—quite conservative. If hiding the monitoring in the background is not feasible or not sufficiently effective, the FTL can also monitor the errors only every several erase cycles. Accordingly, we evaluated how the lifetime improvement is affected by a limited monitoring frequency and observed that a monitoring frequency of 20% (i.e., blocks are monitored once every five P/E cycles) provides sufficient information to sustain the same lifetime extension than full monitoring. In substance, while the process of identifying the weakest pages could at worst require one page read per page written, simple

	Plan 0 (ρ_0=60%)		Plan 1 (ρ_1=75%)		Plan 2 (ρ_2=90%)	
	4000 cycles		2000 cycles		2000 cycles	
Page #	Half rel.	Full rel.	Half rel.	Full rel.	Half rel.	Full rel.
0	-	-	-	-	-	-
1	-	-	40%	-	60%	40%
2	-	-	-	-	-	-
3	30%	-	-	100%	60%	40%
4	-	-	-	-	-	-
5	-	100%	-	100%	60%	40%
6	-	-	-	-	-	-
7	-	-	-	-	-	-
8	-	-	-	-	-	-
9	-	-	30%	-	60%	40%
10	-	-	-	-	-	-
11	-	-	-	-	100%	-
12	-	-	-	-	-	-
13	90%	10%	-	100%	60%	40%
14	-	-	-	-	-	-
15	-	-	-	-	-	-

Figure 8: **Example of a relief plan.** The relief plan is actually made of several plans, each valid for a given amount of relief cycles. According to this plan, blocks will follow Plan 0 during the first 4000 relief cycles then move on to Plan 1 for the next 2000 relief cycles and so on. A plan provides for each page its probability to be relieved. In the example, page 5 is the weakest page and is relieved to the maximum in Plan 0 and Plan 1.

techniques can reduce this overhead to negligible levels without a loss in the effectiveness of the idea.

5.3 Relief Planning Ahead of Time

The *reactive* approach requires to identify the weakest pages during operation and while significant deterioration has already occurred, which somehow limits the potential for relief. More efficient would be to relieve the weakest pages from the very first writes to the device. Interestingly, previous work observed noticeable BER correlation with the page number [7, 3]. Similarly, we observe on our chips a significant correlation between a page position in a block and its endurance. This correlation is important enough to allow us to rank every page per endurance. Thereby, we developed a *proactive* technique to exploit the relief potential more efficiently.

The *proactive* technique requires first a small analysis of the flash chip that we consider. We must characterize the endurance of LSB/MSB page pairs in every position in a block, for a given BER. For each page pair, only the shorter page endurance is considered. This information can be extracted from a relatively small set of blocks (e.g., 10 blocks). Thanks to this information, we will be able to rank the page pairs by their endurance and know which page should be relieved the most. Yet, building an efficient relief plan would also require the knowledge of how many times a block will be allocated to the hot partition during its lifetime, which corresponds to the amount of opportunities to relieve its weakest pages. With this information, one could evaluate to what extent the weakest page of a block can be relieved and how many times the other pages should be relieved to meet the same extended endurance. However, in practice, one cannot have this information ahead of time. Instead, we prepare a sequence of plans targeting increasing hot allocation counts; Figure 8 gives an example of such a sequence. In this example, Plan 0 contains the relief information for the first 4000 relief cycles. Once a block has been allocated to the hot partition 4000 times, one moves to Plan 1 for the next 2000 relief cycles. The entries in the plans are probabilities for a page to be either fully relieved, half relieved, or normally programmed. Hence, when a block is allocated to the hot partition, before programming a page, one should first consult the plan and decide whether or not the current page should be skipped.

To create such plans, sequentially starting from Plan 0, we first refer to the page pairs endurance analysis to identify the weakest pair position w. Each Plan p is built assuming an intermediate hot allocation ratio ρ_p (e.g., 60% for Plan 0) that grows from one plan to the next. The higher it is, the more flexible the plan will be and applications with large hot ratios will largely benefit from half relief cycles, while applications with low hot ratios will not be relieved as aggressively as they should. After choosing a ratio, we evaluate the maximum possible endurance extension with full relief for the weakest page pair w, $E_{T,p} = E_{X,w}(\rho_p, \alpha_F)$. The expected number of relief cycles for this Plan p is thus $L_p = \rho_p \cdot E_{X,w}$ minus the total length of the previous plans. Hence in the example, the hot allocation ratio ρ_1 of Plan 1 would provide 2000 more relief cycle than Plan 0. Thereby, when a block exceeds 4000 relief cycles before turning bad, it means that the actual ρ is larger than ρ_0 and the block should move on to the next plan, which targets a higher ρ.

Once the target endurance is set, for every page pair i having an endurance E_i lower than $E_{T,p}$, we compute the number of relief cycles R_i that would be required for them to align their endurance to $E_{T,p}$. Setting

$$E_{X,i}(\rho_i, \alpha) = \frac{E_i}{(1-\rho_i)+\rho_i\alpha} = E_T \qquad (4)$$

and considering that $\rho_i = R_i/E_T$, we simply obtain

$$R_i = \frac{E_T - E_i}{1-\alpha}. \qquad (5)$$

Here, α is the fraction of stress corresponding to half or full relief cycles, or to a combination of the two, and we still need to decide which type of relief to use.

As discussed in Section 4.2, half relief is most efficient in terms of avoided stress per written data and in terms of performance, and, hence, we will maximize its usage. For every page i to be relieved, we evaluate with Equation (5) and $\alpha = \alpha_H$ the number of half relief cycles that

would be necessary to reach the endurance $E_{T,p}$. If the required number of half relief cycles is larger than the number of relief cycles in this plan L_p, we are forced to consider some full relief as well. Trivially, from Equation (5) and with $L_p = R_i$, we determine the fraction λ of full relief cycles such that the average fraction of stress is

$$\alpha = \lambda \alpha_F + (1-\lambda)\alpha_H = 1 - \frac{E_T - E_i}{L_p}. \quad (6)$$

To construct Plan $p+1$, every page that was relieved, even partially, according to Plan p will be set to the maximum relief rate (i.e., 100% full relief), and the above process is repeated.

Similarly to the *reactive* approach, we restrict to r the maximum number of relieved pages in order to limit the potential performance drop. For the *proactive* technique, we can solely evaluate what would be the average number of pages relieved per plan by summing every page probability to get relieved. For example, in Figure 8, for Plan 0 the average number of relieved pages is $2 \cdot (1+0.1) + 0.3 + 0.9 = 3.4$ pages out of 32 (remember that a full relief skips two pages). Limiting the average number of pages relieved will at some point bound the target endurance. This is illustrated in Figure 8 with Plan 2. Assuming that a maximum of eight pages on average is allowed, the original $E_{T,2}$ would have required the number of relieved pages to be larger than this. Hence the $E_{T,2}$ is reduced to meet the requirements, which reduces the relief rate of every page to meet the average of eight relieved pages per cycle. The plan that requires to reduce its original target endurance becomes the latest plan. Once a block completed this last plan, it will simply stop having to relieve any page until the end of its lifetime.

This technique requires to store the plans in the FTL memory. Each plan has two entries for each LSB/MSB pair and each entry can be encoded on 8 or 16 bits, depending on the desired precision, resulting in 256–512 Bytes per plan, which is negligible for most environments. Besides, the tables are largely sparse and could be further reduced by means of classical compression strategies (e.g., hash tables) to fit in memory sensitive environments.

6 Experiments and Results

We evaluate here the expected lifetime extension achievable with the two relief strategies presented. In the next sections, we explain how we begin by combining error traces acquired from real NAND flash chips with simulation to obtain a first assessment of the improvements of block endurance and, consequently, of device lifetime. We then refine our experimental methodology by implementing a trace-driven simulator and a couple of state-of-the-art FTLs, and by evaluating more accurately the impact of our technique. We use a number of benchmarks to show not only the lifetime improvement but also the minimal effect (often favorable) of our technique on execution time.

6.1 Collecting Traces and Simulating Wear

To assess the impact of our technique, we first collected real error traces from 100 blocks from each of our chips that went through thousands of regular P/E cycles; we collected the error count of every page at every P/E cycle. We then used the collected traces to simulate what would happen of the blocks when going through P/E cycles during normal use of the device. At each simulated P/E cycle, each block is either allocated to the hot partition (i.e., where pages can be relieved) or to the cold one, depending on a hot-write probability; this parameter simulates the behaviour of an FTL and defines the probability for a block to be allocated to the hot partition. When a block is allocated to the cold partition, a normal P/E cycle occurs: every page is considered programmed. When a block is allocated to the hot partition, the weak pages are relieved instead. The *reactive* approach uses the error counts to determine pages as weak if they have reached the predefined threshold k. The *proactive* approach, on the other hand, relies solely on the relief plans prepared in advance to determine the weak pages to be relieved. While we simulate successive writes to the device, we count how many times each page has been written and to what extent it has been relieved. Whenever our real traces tell us that one page of a block has reached a given BER, considered as the maximum correctable BER, we render the block as bad and stop using it. At the end, the simulator reports the total amount of data that could be written in each block—that is, the lifetime of the block under a realistic usage of the device.

6.2 Block Lifetime Extension

We use our wear simulation method to first evaluate the lifetime enhancement provided by our techniques at the block level. In this context, we consider a block to be bad as soon as one of its pages reaches the given BER. Considering a 60% hot write ratio, Figure 9 shows the lifetime of every block for both our flash chips assuming a maximum BER of 10^{-4}; it compares our *proactive* and *reactive* techniques to the baseline. The blocks are ordered on the x-axis with the one with the lowest lifetime on the left up to the one with the largest on the right. The bottom curve is the lifetime of each block when stressed normally, while the two curves on the top corresponds to the lifetime when applying our techniques. The relief effectiveness varies depending on the actual block,

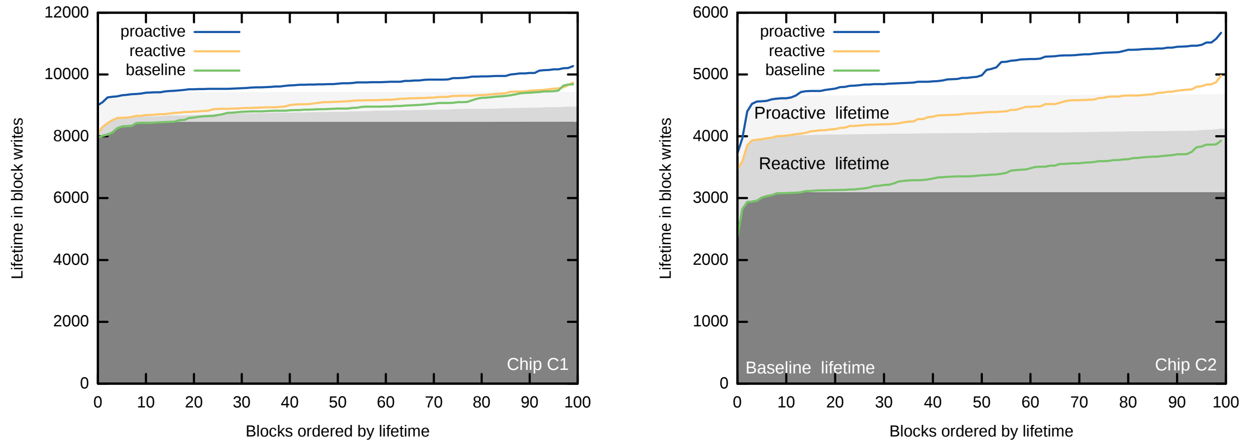

Figure 9: **Block lifetime improvement.** The curves show the individual block lifetime, and the surface areas the device lifetime, assuming it can cumulate up to 10% bad blocks. As expected, the *proactive* technique is more efficient than the *reactive* one. Chip C1 has a relatively small page endurance variance, which limits the efficiency of the *proactive* approach to 10% lifetime extension. Comparatively, C2 offers more room to exploit the relief mechanism and allows the *proactive* approach to extend by 50% the lifetime. For these graphs, we assume a limit BER of 10^{-4} as well as a 60% write frequency to the hot partition.

thereby the block ordering for the two curves is not necessarily the same. The *proactive* approach is more efficient, as it starts relieving pages much sooner than the *reactive* approach. Yet, we believe that there is room to improve our simple weak-page detection heuristic in order to act sooner and be more efficient. Chip C1 shows a relatively small page endurance variance, which limits our techniques potential with a lifetime improvement of 10% maximum. This confirms the intuition that a larger page endurance variability and a greater number of pages per block (double for C2 compared to C1) increase the benefit of the presented techniques. In the next section, we translate the block lifetime extension into a device lifetime extension.

6.3 Device Lifetime Extension

We now evaluate the lifetime extension for a set of blocks when relieving the weakest pages. The three grey areas of Figure 9 represent the total amount of data we could write the device during its lifetime using the baseline and our relief techniques. Assuming that the device dies whenever 10% of its blocks turn bad, the ratio of a relief gray area with the baseline area represents the additional fraction of data that we could write: for C2, our *reactive* and *proactive* techniques show a lifetime improvement of more than 30% and 50%, respectively. These results are obtained from a sample of 100 blocks, which are enough to provide an error margin of less than 3% for a 95% confidence level. From this figure, we can also make a quantitative comparison between the error rate leveling

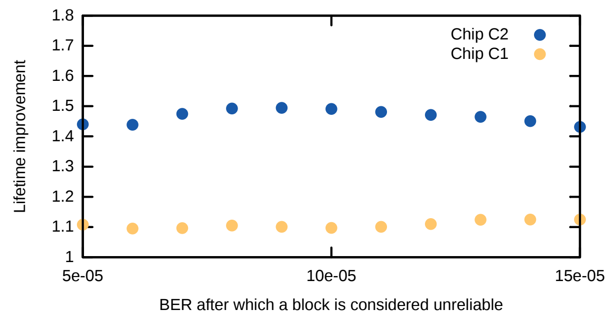

Figure 10: **Lifetime improvement w.r.t. BER threshold.** The BER threshold that indicates when a block is considered unreliable directly affects a device lifetime. Large BER thresholds increase the baseline lifetime and remove room to improvement at the cost of a more expensive ECC.

technique proposed by Pan et al. [20]. If we were to perfectly predict the endurance of every block, we would have a device lifetime that is equal to each individual block lifetime and which corresponds to the total area below the baseline curve. Accordingly, we would get an extra lifetime of 5% and 11% for C1 and C2, respectively, which is an optimistic estimate, yet significantly lower than what the *proactive* approach can bring.

We performed a sensitivity analysis on several parameters that might have an effect on the lifetime extension. For the following results, we focus on the *proactive* strategy. The proportion of bad blocks tolerated by a device had negligible effect on the lifetime extension. As for the

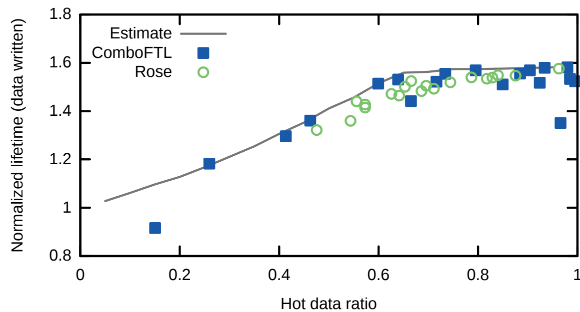

Figure 11: **Lifetime improvement w.r.t. hot write ratio.** The curve gives the expected lifetime extension provided by the *proactive* technique on chip C2. The data points represent results from benchmarks using two different FTLs. Those measurements take into account the writes overhead caused by the hot partition capacity loss. Apart from a couple of outliers, the results are consistent with our expectations.

BER threshold, the effect on lifetime extension is moderate, as illustrated in Figure 10. A larger BER gives more time to benefit from relieving pages, but it also increases the reference lifetime and makes the relative improvement smaller. Finally, the hot write ratio sets by how much our technique can be exploited and has a significant effect on the lifetime extension. The curve labeled "Estimate" in Figure 11 shows the lifetime of a device implementing the *proactive* technique (normalized to the baseline lifetime) as a function of the hot write ratio. We clearly see that the more writes are directed to the hot partition, the better the relief properties can be exploited, as one would expect. The data points on the figure represent the normalized lifetime extension when considering the actual execution of a set of benchmarks with real FTLs, which will be introduced in the next section; these measurements take into account all possible overheads derived from the implementation of the relief technique and match well the simpler estimate. All results show significant lifetime extensions for hot write ratios larger than 40% which is, in fact, in the range where most benchmarks (with very rare exceptions) are in practice.

6.4 Lifetime and Performance Evaluation

The temporary capacity reduction in the hot partition produced by relieving pages decreases its efficiency and is very likely to trigger more often the garbage collector. This effect is more critical for hybrid mapping FTLs that rely on block-level mapping for the cold partition: these FTLs will need to write a whole block even when a single page needs to be evicted from the page-level mapped hot partition (buffer partition) to the block-level mapped cold partition. To refine our estimations and understand the impact on performance, we developed a trace-driven flash simulator and implemented two hybrid FTLs, namely ComboFTL [9] and ROSE [5]. Both FTLs have a hot partition that is mapped to the page level, however their cold partitions are mapped differently. ROSE maps its cold data at the block level, while ComboFTL divide its cold partition in sets of blocks, each being mapped at the page level. Additionally, ComboFTL has a warm partition; we will consider this third partition hot as well, in the sense that pages of blocks allocated to the warm partition will be subject to relief cycles when appropriate. Thanks to the block level mapping, ROSE requires significantly less memory than ComboFTL to be implemented but pays the cost with an execution time 25% larger and a 20% smaller lifetime in average.

In our experimental setup, we assume a hot partition allocating 5% of the total device size and we limit the maximum ratio of relieved pages to 25%, which represents a maximal loss of 1.25% of the total device capacity. Hence, the page relief cost can either be considered as extra capacity requirement (1.25% here) or in a garbage collection overhead that we will now evaluate for two different FTLs.

We selected a large set of disk traces to be executed by both FTLs. First the trace *homesrv* is a disk trace that we collected during eight days on a small Linux home server hosting various services (e.g., mail, file server, web server). The traces *fin1* and *fin2* [2] are gathered from OLTP applications running at two large financial institutions. Lastly, we selected 15 traces that have a significant amount of writes from the MSR Cambridge traces [19]. In our simulation, we assume a total capacity of 16 GBytes and a flash device with the characteristics of C2 (see Table 1). While most of the traces were acquired on disks of a larger capacity, their footprint are all smaller and by considering only the referenced logical blocks (2 MBytes for C2), every selected benchmark fitted in the simulated disk. Importantly, when simulating a smaller device, the hot partition size gets proportionally scaled down, which effectively reduces the hot write ratio and the potential of our approaches and renders the following results conservative.

For the experiments, we considered again a maximum BER of 10^{-4} and a bad blocks limit of 10%. We report in Figure 12 the performance and lifetime results for both chips and of both FTLs executing all the benchmarks with the *proactive* technique. The results are normalized to their baseline counterpart, that is implementing the same FTL without relieving weak pages. (Note that this makes the results for ComboFTL and ROSE not comparable between themselves, but our purpose here is not to compare different FTLs but rather to show that, ir-

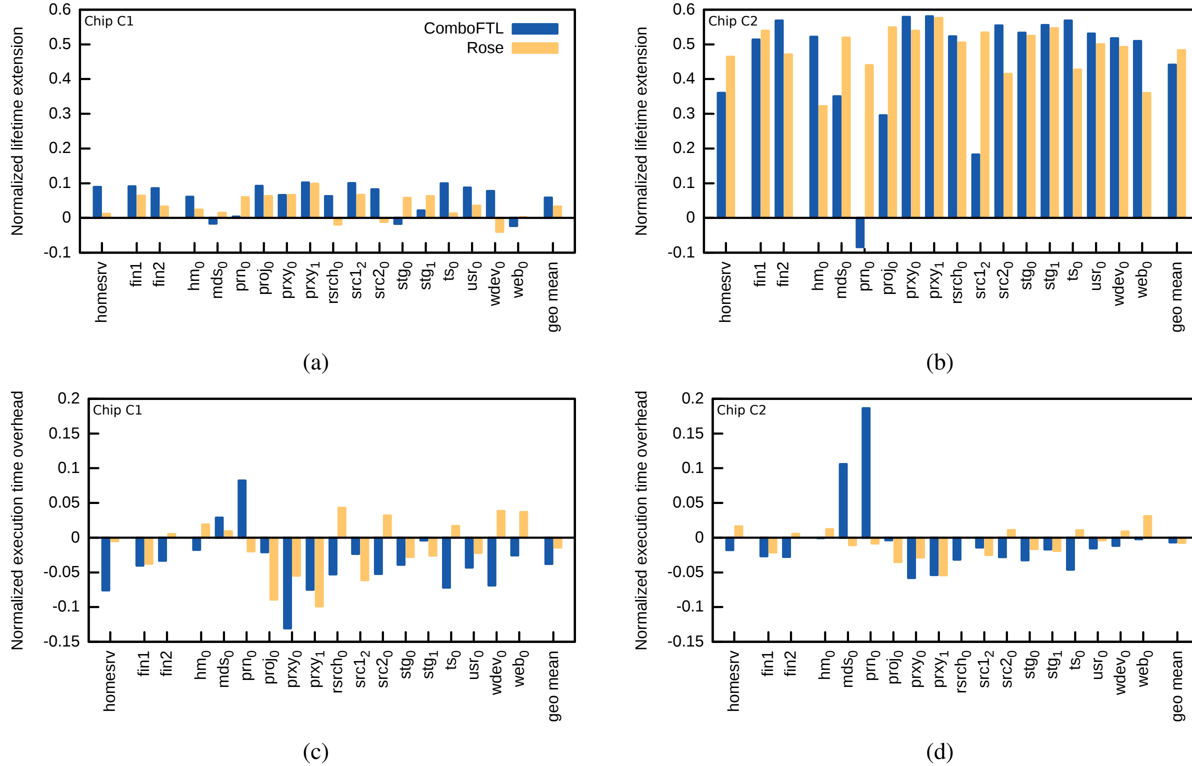

Figure 12: **Performance and lifetime evaluation of our *proactive* technique for various benchmarks running on both chips.** (a) Our relief technique gets at most 10% lifetime extension for the chip C1, (b) whereas for C2 it gives regularly an extra 50% lifetime, but for rare exceptions. In (c) and (d), we see that the execution time is stable for most of the benchmarks despite the capacity loss in the hot buffer. Thanks to the half relief efficiency, several benchmarks even sport a better performance.

respective of the particular FTL, our technique remains perfectly effective). Most of the benchmarks result in a hot write ratio larger than 50% and show a lifetime extension between 30% and 60% for C2. In particular, we observed that ComboFTL frequently fails to correctly identify hot data from the prn_0 trace; this results in a large amount of garbage collection, a poor hot data ratio, and a performance drop of 20% when relieving weak pages—ROSE performs significantly better here. Overall, despite this pathological case, the *proactive* relief technique brings an average lifetime extension of 45% and a execution time *improvement* within 1%. The execution time improvement comes thanks to the half relief efficiency, which provides significantly smaller write latencies. In summary, the *proactive* approach provides a significant lifetime extension with a stable performance and a negligible memory overhead.

7 Conclusion

In this paper, we exploit large variations in cell quality and sensitivity occurring in modern flash devices to extend the device lifetime. We better exploit the endurance of the strongest cells by putting more stress on them while periodically relieving the weakest ones of their duty. This gain comes at a moderate cost in memory requirements and without any loss in performance. The proposed techniques are a first attempt to benefit from page-relief mechanisms. While we already show a lifetime improvement of up to 60% at practically no cost, we believe that further investigation of the effects of our method on data retention as well as research on other wear unleveling techniques could help to further balance the endurance of every page and block. In future flash technology nodes, process variations will only become more critical and we are convinced that techniques such as the ones presented here could help overcome the upcoming challenges.

References

[1] AUCLAIR, D., CRAIG, J., GUTERMAN, D., MANGAN, J., MEHROTRA, S., AND NORMAN, R. Soft errors handling in EEPROM devices, Aug. 12 1997. US Patent 5,657,332.

[2] BATES, K., AND MCNUTT, B. OLTP application I/O, June 2007. http://traces.cs.umass.edu/index.php/Storage/Storage.

[3] CAI, Y., HARATSCH, E., MUTLU, O., AND MAI, K. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Design, Automation & Test in Europe Conf. & Exhibition* (Dresden, Germany, Mar. 2012), pp. 521–26.

[4] CHANG, L.-P. A hybrid approach to NAND-flash-based solid-state disks. *IEEE Trans. Computers 59*, 10 (Oct. 2010), 1337–49.

[5] CHIAO, M.-L., AND CHANG, D.-W. ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation. *IEEE Trans. Computers 60*, 6 (June 2011), 753–66.

[6] CHO, H., SHIN, D., AND EOM, Y. I. KAST: K-associative sector translation for NAND flash memory in real-time systems. In *Design Automation and Test in Europe* (Nice, France, Apr. 2009), pp. 507–12.

[7] GRUPP, L. M., CAULFIELD, A. M., COBURN, J., SWANSON, S., YAAKOBI, E., SIEGEL, P. H., AND WOLF, J. K. Characterizing flash memory: Anomalies, observations, and applications. In *ACM/IEEE Int. Symp. Microarchitecture* (New York, NY, USA, Dec. 2009), pp. 24–33.

[8] HETZLER, S. R. Flash endurance and retention monitoring. In *Flash Memory Summit* (Santa Clara, CA, USA, Aug. 2013).

[9] IM, S., AND SHIN, D. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *Journal of Systems Architecture 56*, 12 (Dec. 2010), 641–53.

[10] JIMENEZ, X., NOVO, D., AND IENNE, P. Software controlled cell bit-density to improve NAND flash lifetime. In *Design Automation Conf.* (San Francisco, California, USA, June 2012), pp. 229–34.

[11] JIMENEZ, X., NOVO, D., AND IENNE, P. Phœnix: Reviving MLC blocks as SLC to extend NAND flash devices lifetime. In *Design, Automation & Test in Europe Conf. & Exhibition* (Grenoble, France, Mar. 2013), pp. 226–29.

[12] LEE, S., SHIN, D., KIM, Y.-J., AND KIM, J. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. *ACM SIGOPS Operating Systems Review 42*, 6 (Oct. 2008), 36–42.

[13] LEE, S.-W., PARK, D.-J., CHUNG, T.-S., LEE, D.-H., PARK, S., AND SONG, H.-J. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embedded Computing Systems 6*, 3 (July 2007).

[14] LIN, W., AND CHANG, L. Dual greedy: Adaptive garbage collection for page-mapping solid-state disks. In *Design, Automation & Test in Europe Conf. & Exhibition* (Dresden, Germany, Mar. 2012), pp. 117–22.

[15] LIU, R., YANG, C., AND WU, W. Optimizing NAND flash-based SSDs via retention relaxation. *Target 11*, 10 (2012).

[16] LUE, H.-T., DU, P.-Y., CHEN, C.-P., CHEN, W.-C., HSIEH, C.-C., HSIAO, Y.-H., SHIH, Y.-H., AND LU, C.-Y. Radically extending the cycling endurance of flash memory (to >100M cycles) by using built-in thermal annealing to self-heal the stress-induced damage. In *IEEE Int. Electron Devices Meeting* (San Francisco, California, USA, Dec. 2012), pp. 9.1.1–4.

[17] MICHELONI, R., CRIPPA, L., AND MARELLI, A. *Inside NAND Flash Memories*. Springer, 2010.

[18] MOHAN, V., SIDDIQUA, T., GURUMURTHI, S., AND STAN, M. R. How I learned to stop worrying and love flash endurance. In *Proc. USENIX Conf. Hot Topics in Storage and File Systems* (Boston, Massachusetts, USA, June 2010).

[19] NARAYANAN, D., DONNELLY, A., AND ROWSTRON, A. Write off-loading: Practical power management for enterprise storage. In *Proc. USENIX Conf. File and Storage Technologies* (San Jose, California, USA, Feb. 2008), pp. 253–67.

[20] PAN, Y., DONG, G., AND ZHANG, T. Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes. *IEEE Trans. Very Large Scale Integration Systems 21*, 7 (July 2013), 1350–54.

[21] PARK, D., DEBNATH, B., NAM, Y., DU, D. H. C., KIM, Y., AND KIM, Y. HotDataTrap: a sampling-based hot data identification scheme for flash memory. In *ACM Int. Symp. Applied Computing* (Riva del Garda, Italy, Mar. 2012), pp. 1610–17.

[22] PARK, J.-W., PARK, S.-H., WEEMS, C. C., AND KIM, S.-D. A hybrid flash translation layer design for SLC-MLC flash memory based multibank solid state disk. *Microprocessors & Microsystems 35*, 1 (Feb. 2011), 48–59.

[23] SCHWARZ, T., XIN, Q., MILLER, E., LONG, D. D. E., HOSPODOR, A., AND NG, S. Disk scrubbing in large archival storage systems. In *IEEE Int. Symp. Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Volendam, Netherlands, Oct. 2004), pp. 409–18.

[24] WANG, C., AND WONG, W.-F. Extending the lifetime of NAND flash memory by salvaging bad blocks. In *Design, Automation & Test in Europe Conf. & Exhibition* (Dresden, Germany, Mar. 2012), pp. 260–63.

[25] WU, M., AND ZWAENEPOEL, W. eNVy: a non-volatile, main memory storage system. In *Sixth Int. Conf. on Architectural Support for Programming Languages and Operating Systems* (San Jose, California, USA, Oct. 1994), pp. 86–97.

[26] ZAMBELLI, C., INDACO, M., FABIANO, M., DI CARLO, S., PRINETTO, P., OLIVO, P., AND BERTOZZI, D. A cross-layer approach for new reliability-performance trade-offs in MLC NAND flash memories. In *Design, Automation & Test in Europe Conf. & Exhibition* (Dresden, Germany, 2012), pp. 881–86.