A (Nearly) Free Lunch: Extending NAND Flash Lifetime by Exploiting Neglected Physical Properties

THÈSE N^{O} 6388 (2014)

PRÉSENTÉE LE 10 DÉCEMBRE 2014

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE D'ARCHITECTURE DE PROCESSEURS

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Xavier JIMENEZ

acceptée sur proposition du jury:

Prof. M. Oderesky, président du jury
Prof. P. Ienne, directeur de thèse
Prof. P. Desnoyers, rapporteur
Prof. B. Falsafi, rapporteur
Prof. S. H. Noh, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2014

Acknowledgements

First of all I would like to express my sincere gratitude to my advisor Prof. Paolo Ienne for his patience, enthusiasm, guidance, and continuous support. Paolo, I am particularly grateful for the trust you put in me and the freedom you gave me throughout my research.

I would also like to thank my thesis committee: Prof. Babak Falsafi, Prof. Peter Desnoyers, and Prof. Sam H. Noh, for their insightful comments and feedback.

My sincere thanks also goes to Dr. David Novo for his invaluable support. Thank you David for all these week-ends and nights spent on improving the writing and flow of my papers. You taught me a lot.

I would like to thank the rest of the *LAPiens*, Chantal Schneeberger, René Beuchat, André Guignard, Prof. Philip Brisk, Dr. Madhura Purnaprajna, Dr. Robert Fasthuber, Dr. Theo Kluter, Dr. Hadi Parandeh-Afshar, Dr. Ali Galip Bayrak, Andrew Becker, Nithin George, Ana Petkovska and Grace Zgheib for the great moments spent at the lab and during our outings. Chantal, thank you for always being cheerful, I will never forget your recognizable laughs. René, André, and Theo thank you for sharing your valuable experience with me.

A special thanks to my colleagues and friends Dr. Romain Rossier, Dr. Thomas Bugnon, Dr. Basile Schaeli, Dr. Florent Garcin, Dr. Fabrice Rousselle, Dr. Rafik Chaabouni, Dr. Julien Andrès, Vahid Babaei, Sami Arpa, Prof. Mike Ferdman, et al. for all the great time spent together at lunch or during dinner parties. Our joyful discussions and board or card game sessions never failed to cheer me up during rough times.

I would like to thank my parents and family for their infinite support and love. Thank you Mom and Dad for providing me with all the resources you could give in order to let me chose my path. Thank you for your trust and care, I am very proud to be your son.

Last but not the least, I would like to thank my wife Audrey for her patience and for the two beautiful kids she gave me during my PhD, Ewan and Alix, certainly my best contributions. Audrey, I know that it was not always easy to get along with my PhD, particularly with two energetic kids at home. I am infinitely grateful for your understanding, courage and love. I could not dream of a better wife. I love you.

Abstract

NAND flash is a key storage technology in modern computing systems. Without it, many devices would probably not exist today or would at least not benefit from as many features. The very large success of this technology motivated massive efforts to scale it down in order to increase its density further. However, NAND flash is currently facing physical limitations that prevent it reaching smaller cell sizes without severely reducing its storage reliability and lifetime. Accordingly, in the present thesis we aim at relieving some constraints from device manufacturing by addressing flash irregularities at a higher level. For example, we acknowledge the fact that process variation plus other factors render some regions of a flash device more sensitive than others. This difference usually leads to sensitive regions exhausting their lifetime early, which then causes the device to become unusable, while the rest of the device is still healthy, yet not exploitable. Consequently, we propose to postpone this exhaustion point with new strategies that require minimal resources to be implemented and effectively extend flash devices lifetime. Sometimes, our strategies involve unconventional methods to access the flash that are not supported by specification document and, therefore, should not be used lightly. Hence, we also present thorough characterization experiments on actual NAND flash chips to validate these methods and model their effect on a flash device. Finally, we evaluate the performance of our methods by implementing a trace-driven flash device simulator and execute a large set of realistic disk traces. Overall, we exploit properties that are either neglected or not understood to propose methods that are nearly free to implement and systematically extend NAND flash lifetime. We are convinced that future NAND flash architectures will regularly bring radical physical changes, which will inevitably come together with a new set of physical properties to investigate and to exploit.

Keywords: *NAND flash, SLC, MLC, Characterization, Experiment, Endurance, Lifetime, Performance, Reliability, Wear-leveling.*

Résumé

La mémoire flash NAND est une technologie de stockage clef dans les systèmes informatiques modernes. Sans elle, de nombreux types de dispositifs n'existeraient probablement pas aujourd'hui ou ne bénéficierait pas d'autant de fonctionnalités. Le grand succès de cette technologie a motivé des efforts considérables vers sa miniaturisation afin d'en augmenter plus encore sa densité. Toutefois, la mémoire flash NAND est actuellement confrontée à des limitations physiques qui l'empêchent d'aller vers des tailles de cellules plus petites sans que cela ne dégrade fortement sa fiabilité de stockage et sa durée de vie. Ainsi, dans la présente thèse, nous visons à soulager les contraintes liées à l'élaboration de cette mémoire, en exploitant certaines irrégularités de la mémoire flash depuis un niveau supérieur. Par exemple, nous observons le fait que la varation du processus de fabrication ainsi que d'autres facteurs rendent certaines régions de la mémoire flash plus sensibles que d'autres. Cette différence conduit généralement à des régions sensibles épuisants leur durée vie trop tôt. Ce qui rend alors la mémoire flash inutilisable, quand bien même le reste de sa mémoire est encore en viable, mais non exploitables. Par conséquent, nous proposons de repousser ce point d'épuisement avec de nouvelles stratégies nécessitant un minimum de ressources à mettre en œuvre et permettant d'étendre efficacement la durée de vie des stockages à base de mémoire flash. Les stratégies que nous proposons impliquent parfois des méthodes non conventionnelles pour accéder à la mémoire flash. Celles-ci ne sont pas prises en charge par les documents de spécification des fabricants et, par conséquent, ne doivent pas être utilisées à la légère. Ainsi, nous présentons également un ensemble d'expériences de caractérisation sur de réelles puces flash NAND pour valider nos méthodes et modéliser leurs effets sur les stockages à base de mémoire flash. Enfin, nous évaluons le rendement de nos méthodes en mettant en place un simulateur de disque flash, à travers lequel nous exécutons un grand nombre de traces de disques acquises sur des systèmes réels. Dans l'ensemble, nous exploitons des propriétés qui sont soit négligées ou encore incomprises pour proposer des méthodes à coût négligeable et montrons systématiquement une extension de durée de vie de la mémoire flash NAND. Nous sommes convaincus que les futures architectures flash NAND apporterons régulièrement des changements radicaux dans le processus de fabrication, ce qui amènera inévitablement un nouvel ensemble de propriétés physiques à investiguer et exploiter.

Mots-clefs : *NAND flash, SLC, MLC, Characterization, Experiment, Endurance, Lifetime, Performance, Reliability, Wear-leveling.*

Contents

Acknowledgements iii

Abstract (English/Français) iv

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Efforts So Far . . . 2
- 1.3 Thesis Contributions . . . 2
- 1.4 Organization of this Thesis . . . 3

2 NAND Flash Memory 5
- 2.1 Storage Mechanism . . . 5
  - 2.1.1 NAND Architecture . . . 5
  - 2.1.2 Programming . . . 6
  - 2.1.3 Reading . . . 7
  - 2.1.4 Erasing . . . 7
- 2.2 Reliability . . . 7
  - 2.2.1 Flash Endurance . . . 8
  - 2.2.2 Error Correcting Codes . . . 8
  - 2.2.3 Data Retention . . . 8
  - 2.2.4 Interferences . . . 9
- 2.3 Multilevel Cells . . . 9
  - 2.3.1 Reading and Programming . . . 9
- 2.4 Flash Translation Layer . . . 10
  - 2.4.1 Wear-Leveling . . . 11
  - 2.4.2 Garbage Collection . . . 12
  - 2.4.3 Address Mapping . . . 12
- 2.5 Conclusion . . . 13

3 Flash Characterization 15
- 3.1 Introduction . . . 15
- 3.2 Measured Responses . . . 16
  - 3.2.1 Access Latency . . . 16
  - 3.2.2 Error Count . . . 17

3.2.3 Energy . . . 18
3.3 Influencing Factors . . . 18
3.3.1 Cell Condition . . . 19
3.3.2 Write Data Pattern . . . 19
3.3.3 Time . . . 21
3.3.4 Temperature . . . 21
3.3.5 Reference Threshold Voltage . . . 22
3.3.6 Physical Cell Position . . . 22
3.4 Experimental Setup . . . 23
3.4.1 Architecture . . . 23
3.4.2 Characterization Procedure . . . 24
3.5 Related Work . . . 25
3.6 Conclusion . . . 26

4 Libra: Balancing Mixed SLC-MLC Wear **27**
4.1 Introduction . . . 27
4.2 SLC-MLC Hybrid Storage . . . 29
4.3 Libra: Soft Partitions to Balance Wear . . . 31
4.3.1 Faster MLC: Managing MLC as SLC . . . 31
4.3.2 Software-Controlled Log Buffer . . . 32
4.3.3 Libra Implementation . . . 34
4.3.4 Libra Lifetime Model . . . 35
4.4 SLC-mode Characterization . . . 36
4.4.1 Considering the Recovery Factor . . . 38
4.4.2 SLC-mode Wear . . . 39
4.5 Results . . . 42
4.5.1 Experimental Setup . . . 42
4.5.2 Soft vs. Hard Partitioned Hybrid FTLs . . . 45
4.5.3 Generalization of Experimental Results . . . 46
4.6 Related Work . . . 48
4.7 Conclusions . . . 49

5 Phœnix: Reviving MLC Blocks as SLC **51**
5.1 Introduction . . . 51
5.2 Reviving Bad Blocks . . . 51
5.2.1 Reviving MLC Blocks in SLC-mode . . . 52
5.3 Device Degradation Models . . . 53
5.3.1 Block Endurance Distribution . . . 54
5.3.2 Analytical Model of Baseline Device Lifetime . . . 55
5.3.3 Analytical Upper Bound of Phœnix Device Lifetime . . . 56
5.4 Results . . . 57
5.5 Future Perspectives . . . 59
5.6 Related Work . . . 60

5.7 Conclusion 61

6 Wear Unleveling: Relieving Weak Pages to Balance Endurance 63
6.1 Introduction 63
6.2 Relieving Pages 64
6.2.1 Definition 64
6.2.2 Page Endurance 65
6.2.3 Understanding the Relieving Effect 66
6.3 Implementation in FTLs 69
6.3.1 A New Page State 69
6.3.2 Mitigating the Capacity Loss 70
6.3.3 Reactive Approach: Identify Weak Pages on the Fly 71
6.3.4 Proactive Approach: Relieving Plan Ahead of Time 73
6.4 Experiments and Results 76
6.4.1 Collecting Real Traces and Simulating Wear 76
6.4.2 Block Lifetime Extension 77
6.4.3 Device Lifetime Extension 77
6.4.4 Lifetime and Performance Evaluation 79
6.5 Related Work 81
6.6 Conclusion 84

7 Conclusions 87

Bibliography 94

Curriculum Vitae 95

1 Introduction

NAND flash memory is currently the densest semiconductor memory technology and therefore the cheapest. Added to that factor, its low power consumption, mobility and high performances makes it remarkably successful for light embedded storage applications, particularly for cases where classic magnetic disks are not adapted. Accordingly, NAND flash memory is today by far the leader in storage media for handheld devices. Furthermore, despite the fact that magnetic disks or even tapes remain more cost effective and reliable than NAND flash for very large storage systems, it is not uncommon for large tiered (i.e., hierarchical) storage to use NAND flash at the highest storage levels to act as a fast and low energy storage cache. Therefore, NAND flash is a particularly important actor in the storage ecosystem and can be found in a large range of applications, such as storage for smartphones, tablets, ultrabooks, mp3 players, removable storage, solid state drives or as large storage caching.

1.1 Motivation

Although NAND flash memory is already well established, manufacturers continue pushing for higher densities in order to provide the most competitive devices. However, during this progression to smaller technology nodes, several unpleasant NAND flash properties start becoming more cumbersome. For example, flash memory cells can only be written a limited number of times before becoming unreliable and this gets fatally more problematic with smaller cell sizes [23]. Consequently, in order to address these issues, flash manufacturers have to put a considerable effort rethinking their low-level cell architecture at every new technology node. However, manufacturers cannot solve all the issues by themselves and must count on the research community that works on high-level strategies designed to delay the flash device wear out as much as possible. In an effort to open new perspectives in that regard, we put a particular effort in this thesis finding new angles to complement these solutions and efficiently help extending the flash device lifetime further.

1.2 Efforts So Far

Solid State Drives (SSDs) manufacturers work hard to bring more intelligence in their flash controllers and design new techniques reducing the amount of data written to their storage. For example, one can write less data by compressing it [59, 42] or detecting redundant chunks of data and *deduplicating* them [14], which tend to increase both the device lifetime and its performances. Thereby, a large set of generic solutions applicable on a wide range of device form factors are proposed improving the control logic with more efficient data mapping strategies [37, 10, 16, 22, 11, 26, 52, 15, 47]. Other techniques are specific to an application. For instance, storage policies and architectures have been design to specifically address flash devices acting as a cache for large storage systems [34, 35, 9, 1, 53, 55].

The limitations of NAND flash inspired other researcher to anticipate similar issues for new emerging non volatile memories, such as *Phase-Change Memory* (PCM), Memristors and *Spin-Transfer Torque memory* (STT-RAM). Currently, the most promising emerging technology is the *Phase-Change Memory* (PCM), which Micron produced in relatively large volumes. However, Micron interrupted their PCM production in the beginning of 2014 to focus on the development of new NAND flash architectures. This fact confirms that NAND flash memory will stay an important actor in the years to come.

1.3 Thesis Contributions

Throughout this thesis, we will reveal a set of physical flash properties neglected by manufacturers' specification documents. From these, we could design original approaches to manage flash devices that are nearly free to implement and contribute to extend their lifetime. Rather than trying to improve existing policies, we propose strategies acting from new angles and being as much as possible complementary to traditional existing techniques. Our methods are aimed to be implemented into the flash memory controller that sits between the host file system and the flash chips themselves. Yet, there are a large set of flash controllers with different processing power as well as resources available, which depend on the storage form factor or target application. Consequently, we designed our methods to be as light as possible to implement in order for them to be applicable by the largest set of controller types. A particularity of these methods is that they often have to break the conventions set by flash manufacturers. Accordingly, in this thesis we will present a set of well-designed experiments to characterize properly their effects on the device. These experiments will serve validating the approaches that we propose as well as modeling and quantifying their effects on flash memory storage devices.

1.4 Organization of this Thesis

In the next chapter, we provide a background on NAND flash memory. We discuss its basic storage mechanisms as well as its particular cell organization. We also present all the main features that should be expected of a flash controller.

In Chapter 3, we discuss on the process of characterizing NAND flash memories. We present the various factors that could affect and bias the results of characterization experiments. Finally, we describe the experimental setup that we built to perform such experiments.

In Chapter 4, we present *Libra* [30], a method supporting mixed sources of wear, which allows using different storage modes inherent to flash memory in a flexible way. We will show that this flexibility enables sharing the wear across the flash cells more efficiently and provides up to one order of magnitude more lifetime compared to previous rigid approaches. Furthermore, this method requires negligible extra resources to be implemented.

In Chapter 5, we present Phœnix [29], a method that relies on two physical properties of flash. First is the nonuniformity of cell degradation over the flash device: some will wear out significantly earlier than others. Second is the fact that storing less information in memory cells renders them more reliable. Therefore, when too many cells become unreliable (*die*) within a block of cells, we propose reviving this block by restricting it to store less information, which allows a lifetime extension of up to 17% for the studied NAND flash chip and comes for free. Lastly, we model the relationship between the cell lifetime variance and Phœnix potential, which let us envision greater lifetime extension with future flash technology nodes.

We go further in Chapter 6 by addressing this lifetime variance on a smaller granularity than Phœnix. Usually, all cells being part of the same block are written together. Yet, we propose to relieve the weakest ones in order to balance the lifetime within a block [31]. This approach breaks the conventional ways of accessing flash and, therefore, requires careful characterization to understand all its effects. We propose two different strategies relieving weak cells and show that for the considered NAND flash chips, up to 60% lifetime extension can be achieved for a minimal cost.

Finally, we conclude this thesis in Chapter 7.

2 NAND Flash Memory

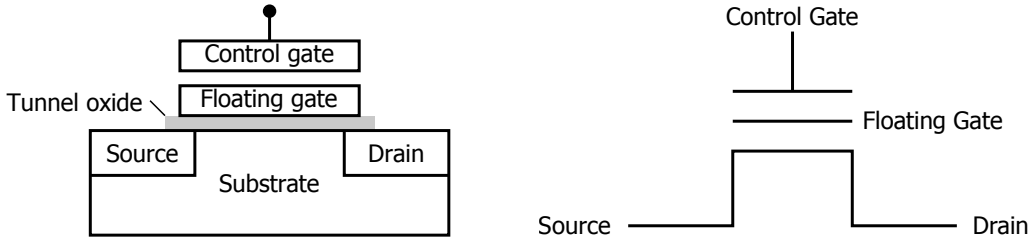
This chapter provides a background on NAND flash memory. It discusses its specific architecture and the peculiar storage mechanisms involved with it. The chapter concludes with a description and comparison of the main flash controller classes.

2.1 Storage Mechanism

Flash memories store information by using electron tunneling to place and remove charges into *floating gates*. Figure 2.1 illustrates the flash cell structure consisting of a MOS transistor made of two gates instead of one. The floating gate in the middle serves as a recipient for electrons. The action of adding electrons into a cell is called *programming*, whereas the removal of this charge is called *erasing*.

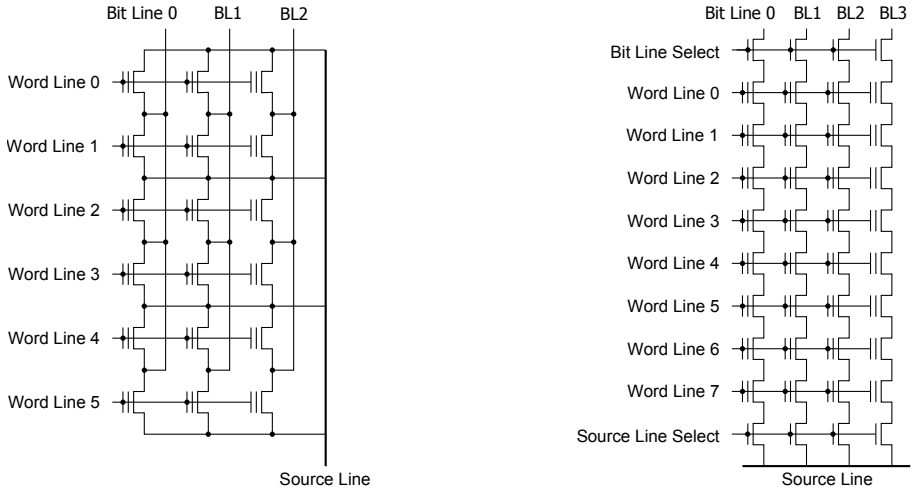
2.1.1 NAND Architecture

Flash memory comes in two main architecture variants: NOR and NAND, illustrated in Figure 2.2. In NOR flash, cells are connected to the bit line in parallel, which resemble a NOR gate: whenever a word line is brought high, the corresponding bit lines will be pulled down. NOR flash is relatively slow to program but allows fast random reads; thereby, it is mainly used to store devices' firmware or BIOS. In contrast, NAND flash has its cells arranged serially in a NAND gate fashion: the bit line is pulled down only when every word line is brought high. This serial structure brings more density (hence, reduced cost) but increases significantly the read latency. This latency increase is somewhat compensated by enlarging the access granularity to a page level (i.e., typically 4–32 kB) instead of a single byte and allows for larger bandwidth. In summary, compared to NOR flash, NAND flash features slower reads, larger write bandwidth and is cheaper than NOR flash. Furthermore, NAND flash success puts significant pressure on its development and production, which results in a highly optimized technology being more advanced than NOR (e.g., smaller feature size). In this thesis, we will focus on the



(a) Floating memory cell (b) Symbolic representation

Figure 2.1: **Flash cell structure.** A flash cell consists of a MOS transistor built with two gates on top of each other instead of one. The gate in the middle is called the *floating gate*.



(a) NOR flash (b) NAND flash

Figure 2.2: **NOR and NAND flash cell organization.** In NOR flash, cells are organized as in a NOR gate, in parallel. In NAND, cells are organized in series, much like a NAND gate.

NAND variant, for which limited lifetime is a greater concern compared to typical NOR flash use cases.

2.1.2 Programming

Programming cells consists of using the Fowler–Nordheim tunneling effect to inject electrons into the floating gate. This effect occurs by grounding both the source and drain and setting a large voltage V_{PGM} (typically about 20 V) on the control gate. In the NAND architecture context, the selected word line is set at V_{PGM}, while every other word line should be biased with an intermediate voltage $V_{P,PASS}$ (typically 10 V) that lets the current flow in unselected bit lines (pulled high).

The current flowing into the floating gate varies significantly from one cell to another. Consequently, the programming process is divided in multiple program/verify cycles. With this

approach, every cell can be programmed independently. Every cell that accumulates a satisfying amount of charges gets its corresponding bit line deactivated and is stopped of being programmed. The process ends when every cell has been programmed. The programming latency depends heavily on the flash parameters and can vary from 250 μs to 2 ms or more.

2.1.3 Reading

Reading a cell consists of testing the voltage threshold of a cell. An erased cell has a voltage threshold lower than 0 V. Accordingly, a voltage of 0 V on the control gate will activate the cell and let it conduct current. In contrast, a programmed cell has negative charge in its floating gate, which increases the voltage threshold required for the cell to conduct current. Therefore, to read cells, the selected word line is set to 0 V, while the other word lines are set to $V_{R,PASS}$ (typically 5 V) in order to let them conduct current, programmed or not. Thereby, erased cells will let current flow on their corresponding bit line, while programmed cells will not. It typically takes about 50 μs to read a page into the internal buffer of the flash chip.

2.1.4 Erasing

Erasing removes the charges from the floating gate and takes about 3 ms. This is achieved by putting a large voltage on the substrate (typically 20 V) while pulling down the word lines of the selected block. Any unselected block sharing the same substrate has its word lines left floating and is unaffected. Within a block, it is not possible to ground only a subset of the word line while letting the rest floating. Due to the proximity of the word lines, this would result in a dielectric breakdown. Therefore, in modern NAND flash, erasing can only be done on the granularity of a block, which is somewhat cumbersome.

The only way to remove charges from the floating gates is to erase them. Therefore, updating a single page of a block would require buffering the complete block, erasing the block and programming back the updated data. Obviously, this would be prohibitive both in terms of time and buffer size requirements. Instead, updating pages must be done *out-of-place*, meaning that every updated page should be programmed in another block with free pages. More on this will be covered in Section 2.4.

2.2 Reliability

In this section, we discuss the NAND flash reliability and the main sources of errors that can be observed.

2.2.1 Flash Endurance

Flash cells degrade while accumulating *Program/Erase* (P/E) cycles [8, 5, 43]. This is provoked by the oxide layer accumulating charge holes, which eases the transfer of electrons. Therefore, cells become progressively less efficient in the retention of charges, more sensitive to neighboring disturbances, and consequently, prone to errors. As a result, all flash blocks experience a gradual *Bit Error Rate* (BER) increase with the number of P/E cycles during their life cycle. Accordingly, manufacturers specify a particular block endurance for their device in terms of P/E cycles. Past this point, flash blocks are considered unreliable and their data integrity becomes compromised. However, even if a uniform wear is assumed among all the blocks, a few flash blocks can wear out before the specified device endurance (and reversely). Indeed, blocks do not present the same level of tolerance towards P/E cycles due to process variation and some blocks might become unreliable significantly sooner than others. Accordingly, flash devices generally reserve a set of spare blocks to replace early failing blocks during the device lifetime [44].

2.2.2 Error Correcting Codes

In NAND flash memory, it is frequent for bits to flip. Consequently, *Error-Correcting Codes* (ECCs) are used to correct a limited number of bit errors within flash pages. For this, flash pages are extended with spare bytes that are used to store metadata (e.g., P/E count, address mapping) and the redundant bits necessary to implement the ECC. The ECC computation is generally the responsibility of flash controller, but there are also some flash chips integrating directly some ECC logic. The most common ECCs implemented for NAND flash are BCH, Reed Solomon, and more recently *Low Density Parity-Check* (LDPC) codes. For every new flash technology node, flash cells shrink to smaller sizes and are more sensitive to interferences, resulting in lower data retention properties. Consequently, the ECC strength (i.e., number of errors that can be corrected) that is required to maintain satisfying block endurance increases drastically at every new technology node. However, a stronger ECC grows in size and requires a more complex and longer error decoding process, which degrades the read latency and size advantages of technology scaling. While improving the performances of ECCs [60] and adapting them specifically for NAND flash can directly improve flash longevity and reliability, we believe that complementary alternatives should be investigated, such as the methods that we propose in this thesis.

2.2.3 Data Retention

The charges of a cell leak over time, which degrades the stored data and eventually leads to unrecoverable data loss. Accordingly, manufacturers must specify a minimum data retention time (e.g., one year) together with the endurance (in terms of P/E cycles) to qualify their flash device lifetime. A common approach to prevent this silent and progressive loss of data is to perform data scrubbing: regularly read old data and assess the current error count; when this

count approaches the limits of the ECC unit in use, the data is safely copied elsewhere before it becomes uncorrectable.

2.2.4 Interferences

The charge leakage is not the only source of information loss. Due to the high density of NAND flash, neighboring accesses can interfere with the data stored in a cell. For example, when a word line is read, nearby neighboring word lines get slowly programmed. If too many reads accumulate on the same word line (typically 100,000 times), the direct neighbors risk to accumulate too many charges and lose the stored information. This effect is called read disturb. Similarly, for program disturb, programming a word line interferes with the neighboring word line that is already programmed. In Chapter 3, we will cover some more interference example when describing flash characterization.

2.3 Multilevel Cells

The continuous pressure to improve the density of NAND flash memory brought multi-bits per cell technology. While classical *Single-Level Cell* (SLC) flash stores one bit per cell, *Multi-Level Cell* (MLC) flash stores multiple bits in a single cell. The generic MLC term generally refers to 2-bit per cell. Other densities have multiple naming conventions, sometimes not very well chosen. For example, 3-bit per cell flash is often referred to as *Triple-Level Cell* (TLC) flash, while in fact seven levels in total are required to encode three bits. Another naming convention for MLCs uses X3 and X4 MLC to identify 3-bit and 4-bit per cell, respectively. In this section, we describe the storage mechanisms of MLC and discuss the consequences on performance and reliability.

2.3.1 Reading and Programming

Encoding n bits requires to identify $2^n - 1$ different voltage levels. Thereby, an MLC requires three different voltage thresholds. Supporting more voltage thresholds means that there will be less margin between the voltage levels. Therefore, it will be more likely for bits to flip when interferences occur. Furthermore, these reduced margins will require a more precise programming phase, which will require more time to be executed and degrade performance.

Figure 2.3 illustrates the programming sequence commonly used for MLC. Starting from an erased block, the *Least Significant Bit* (LSB) of every cell is programmed by targeting a single voltage level, which is performed quickly, because this step does not need to be very precise. Then, the *Most Significant Bit* (MSB) of every cell is programmed, which requires reading the current state first (i.e., the LSBs values) and then pushing the cell voltage to either of the three different levels (see solid arrows in the figure). This second programming requires higher precision and it is typically about four to five times longer than the LSB programming [22].

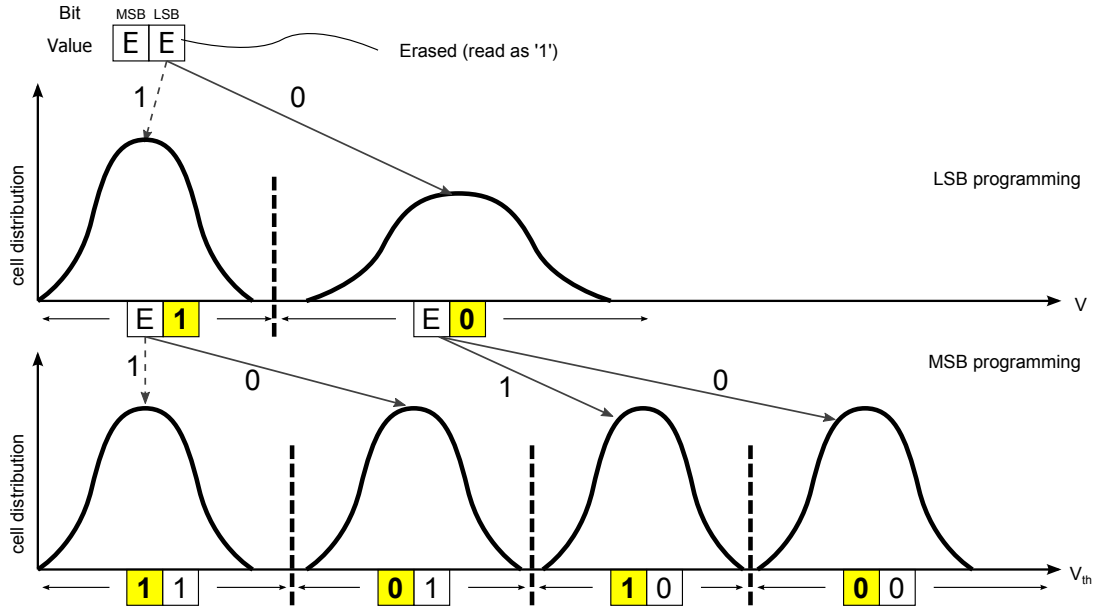


Figure 2.3: **Programming of a 2-bit MLC.** Each bit of a cell is programmed separately. Programming the first bit, or LSB, requires targeting a single level (staying at the erased level is free) and does not need to be very precise. Programming the MSB, requires reading the current state of the cell and targets potentially three different levels, which requires more precision and time.

We also notice in the figure that the bits are not encoded in sequence, but instead use a gray code that prevents a scenario where both cell's bits would flip when shifting from one level to the next one, limiting at the same time the number of errors provoked by such unwanted shifts. In summary, MLC flash brings capacity at the cost of performance and endurance.

2.4 Flash Translation Layer

As discussed in the past sections, NAND flash memory requires extensive maintenance and management to overcome its limitations, such as the out-of-place updates or limited lifetime. In order to address those, an indirection layer, called the *Flash Translation Layers* (FTLs), is placed between the file system and the flash storage. It is typically implemented by the flash controllers within the storage device, although there are a few flash file systems that are able to control directly the NAND flash physical interface and will integrate directly this indirection layer. We will focus here on the FTLs integrated in the device, as it is the most common setup. The FTL maps logical addresses to physical flash locations and must maintain the state of every flash page—typical states are *clean*, *valid*, and *invalid*, as illustrated in Figure 2.4. Valid pages cannot be reprogrammed without being erased, which means that 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 illus-

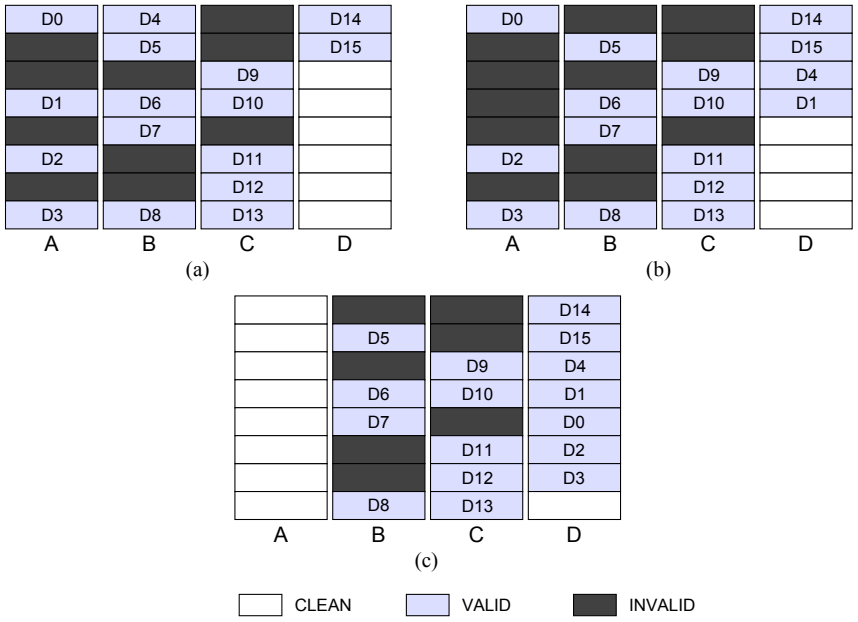


Figure 2.4: **Pages state transitions.** Figure (a) shows the page states generally found in typical flash storage: *clean* when it has been freshly erased, *valid* when it holds valid data, and *invalid* when its data has been updated elsewhere. In Figure (b), data D1 and D4 are updated and their previous values are invalidated from blocks A and B. In Figure (c), block A is reclaimed by the garbage collector. The remaining valid data were first copied to block D, before block A was erased.

trated in Figure 2.4(b), where D1 and D4 have been reallocated. The number of invalid pages grows as the device gets written. At some point, the FTL must trigger the recycling of invalid pages into clean pages. This recycling process is known as garbage collection, which selects a victim block according to a certain policy, copies any remaining valid page to available clean pages, and then erases the victim block. An example of garbage collection is illustrated in Figure 2.4(c), where block A is selected as the victim. Next, we describe in further details the most important tasks deployed by typical FTLs.

2.4.1 Wear-Leveling

FTLs implement several techniques that maximize the use of the blocks' limited endurance to guarantee a sufficient device lifetime. One central approach is to even the wear on every block to prevent a few blocks from getting worn out too rapidly. This is generally performed by wear-leveling, which targets a uniform P/E count on every block of the device [61, 12]. Therefore, the maximum capacity of the device can be guaranteed for a longer time.

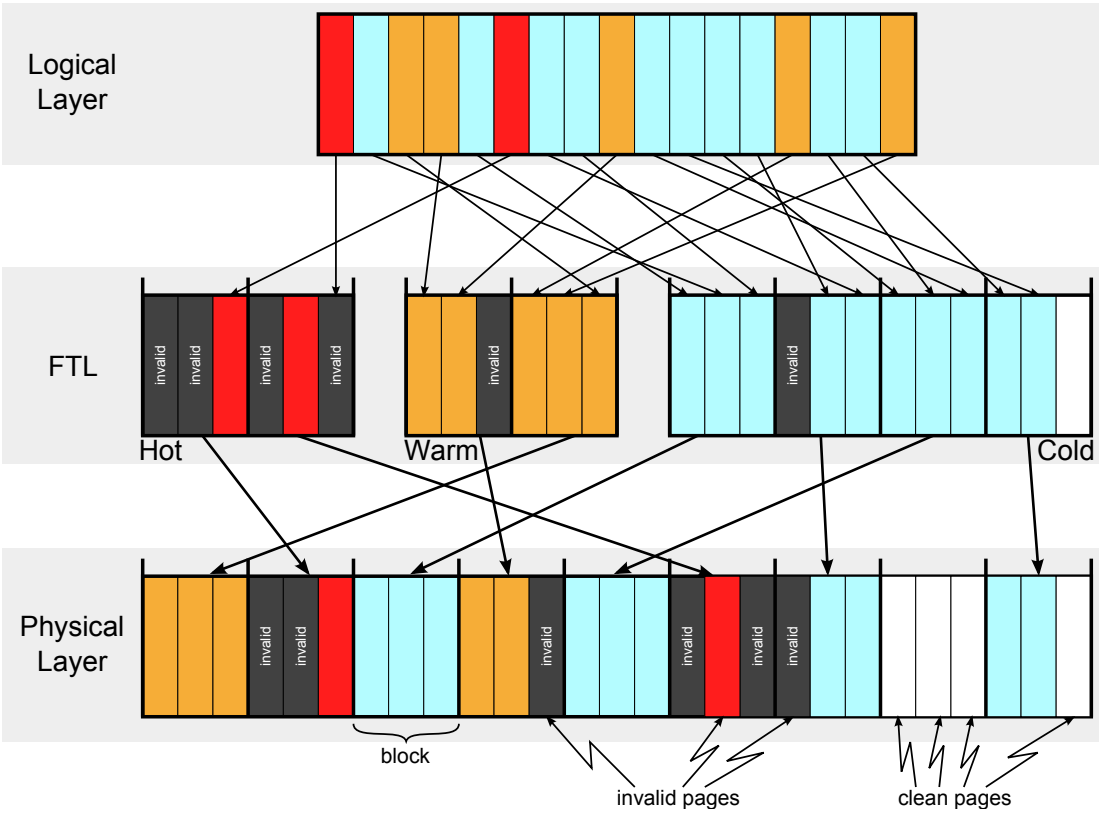


Figure 2.5: **Flash Translation Layer example.** An example of page-level mapping distinguishing update frequencies in three categories: *hot*, *warm* and *cold*. In this thesis, we will often exploit the fact that hot partitions represent a small capacity proportion of the device, while at the same time a significant ratio of writes gets directed to them.

2.4.2 Garbage Collection

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 [61]. *Hot* data have a higher probability of being updated and invalidated soon, resulting in *hot* blocks with a large amount of invalid pages that reduce the garbage collection overhead. Figure 2.5 shows an example FTL that identifies three different *temperatures* (i.e., update frequencies), labeled as *hot*, *warm*, and *cold*. Literature is rich with new heuristics to identify hot data [39, 11, 26, 52, 51].

2.4.3 Address Mapping

FTLs can chose different address mapping granularity. Two granularity types that are straightforward for NAND flash are the page-level and block-level granularity. The block-level map-

ping requires a small mapping table, but generates expensive garbage collection overhead when only a few pages per block must be updated. In contrast, the page-level mapping [24, 41] is much more flexible but requires a large translation table (typically stored in expensive SRAM or DRAM) that is prohibitive for small storage systems with few resources available.

A cost-effective intermediate solution is to use a hybrid mapping. FTLs using this type of mapping are called hybrid FTLs [15, 16, 36, 37, 39]. The device maintains two set of blocks and maps them at a different granularity. A small set of blocks acts as a log buffer that is mapped at the page level. The other set is called the data partition and represents the device capacity. It is mapped at the block level. The purpose is to direct small random writes to the log buffer so that they can be written back to the large data partition in-order as big chunks. Furthermore, it also regroups data that is likely to be overwritten soon into the log buffer, because invalidating a page from the page-level mapped region will generate a significantly lower garbage collection overhead than invalidating a page from a block-level partition. Such an FTL requires a decision level, to decide whether a data write should be directed to the buffer or to the data partition. This decision can for example be taken based on the write request size: relatively large sequential write requests are less likely to be updated in the near future compared to small random writes.

2.5 Conclusion

This chapter has introduced background information on NAND flash technology. Specifically, we compared the NAND and NOR architectures, discussed the mechanisms to read, program and erase flash cells, listed the reliability limitations of this technology, and explained the role of the FTLs. In the next chapter, we will discuss about the flash characterization process and present the experiment platform that we design for it.

3 Flash Characterization

In this chapter, we cover the basic principles for experimenting on NAND flash memory in order to characterize undisclosed properties. We describe the limited set of outputs available from a typical consumer chip and explain how to derive useful information out of it. We also present the experimental setup that we built to perform all the characterizations presented in this thesis.

3.1 Introduction

Characterizing flash devices is a common approach to unveil properties that are typically not published in the manufacturers' specification documents and difficult to predict from the theory or models alone. For example, it can be used to extract statistics on a device performance and quantify temporal or spatial variances. Furthermore, it can be used to observe the effects of original ways to access the flash memory. Indeed, while flash memory is meant to be written sequentially, it can also be used unconventionally, which will often have an influence on its characteristics.

Many previous works proposed new programming schemes while relying on wrong assumptions that can be simply invalidated with a proper characterization of flash devices. A recurrent example is the assumption that the endurance of flash is solely dependent on the number of erases that is performed on a block and not on the data that is programmed in the cells. Accordingly, this led to numerous encoding technique proposals typically inspired from *Write-Once Memories* (WOM) [27, 33, 20, 21], which consists of encoding a set of logical bits on a larger number of cells. For example, a *two write* code would allow storing two bits of information on three cells twice before requiring to erase their corresponding cells, which would increase the effective written bits in the three cells to four bits per P/E cycles. Assuming that the endurance is exclusively related to the P/E cycles, this would trivially bring an endurance improvement. However, as we will discuss in the coming chapters, programming a cell at an intermediate level or at the highest level will generate a significantly different amount of wear and cell-to-cell interference. Specifically, reprogramming a page multiple

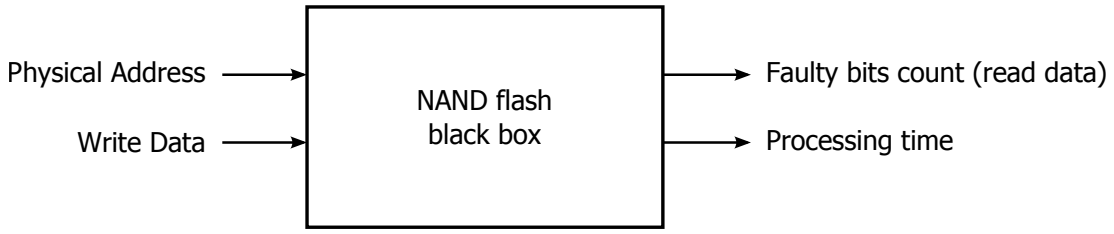


Figure 3.1: **A NAND flash black box model.** The figure illustrates the limited interaction possible with a NAND flash chip.

times will inevitably increase the average programming voltage level of its cells, significantly accelerate the cells degradation, and increase data disturbance, which would severely reduce or even annihilate the benefits originally promised by the proposed encoding. Hence, we believe that characterization is a critical step when designing a storage strategy that is not covered by typical hardware specification documents.

Regarding NAND flash memory consumer chips, only a limited number of responses is accessible, such as the faulty bit count or the access latency. From these, the state of the device can be somehow derived. Figure 3.1 illustrates the black box model of a NAND flash chip. We can input data at a specific physical location and later read back the data and observe how the data was degraded. It can also be useful to monitor the time it takes for each access to be processed. In general, characterization involves cycling the flash device, that is, to program and erase continuously a set of blocks of the flash device and observe how the performance and bit error rate is affected depending on the access patterns. Yet, many factors should be taken into consideration when designing a characterization experiment in order to provide meaningful information, such as the ambient temperature, the cycling frequency, or the data patterns that are programmed. In this chapter, we will present the typical responses that we can measure during the characterization experiments and describe all the factors that we know of and that can have an effect on those responses. In the next section, we start describing the set of response that we can typically extract from a NAND flash chip.

3.2 Measured Responses

The process of characterizing a device consists in sending stimuli to a device, often representing a typical usage environment, while collecting and monitoring the responses or state of the device, which we will discuss in the following subsections.

3.2.1 Access Latency

In NAND flash memory, the program and erase latencies are subject to variations. In contrast, read accesses remain relatively constant throughout the device's lifetime. Furthermore, we can observe that the programming and erasing times can deviate depending on other fac-

tors (e.g., cell condition, ambient temperature) that we will cover in further detail in the next section. One interesting feature is that the more a cell is damaged, the shorter is the programming time [22, 17]. Hence, depending on the time it takes to program a page, it might be possible to evaluate approximately its current health and get a sense of the remaining endurance of a page. As seen in Section 2.3, LSB and MSB programming times are very distinctive. Therefore, the program latency is also a good mean to understand whether we are programming an LSB or MSB page, which is helpful to identify the position of every LSB and MSB page within a block. This helps to recover the internal page mapping, which is not necessarily specified by the manufacturer. Regarding the erase latency, in contrast to the program latency, it increases progressively with cells wearing out. The accumulated charges trapped into the oxide layer make it more difficult for the erase process to achieve a satisfying voltage level.

In order to evaluate the time it takes to perform a programming or erasing command as well as any flash command, we can probe the *busy* signal provided by the ONFI interface. This signal becomes active (low) whenever a command is being processed by the flash memory and is released once the command is processed. Therefore, every time the flash is read, programmed or erased, we can accurately evaluate the time that it takes to perform the operation internally simply by measuring the pulse width of this *busy* signal. It should be noted that advanced features allowing interleaving commands (e.g., caching, multi-banking) hide at least partially the time it takes to perform a single command and will not allow the command time to be evaluated properly. Hence, during our experiments, we enforced regular basic accesses in order to recover the processing time of every access to the flash memory.

3.2.2 Error Count

The voltage threshold of a programmed cell changes over time for multiple reasons. For example, charges regularly leak out of the floating gate, which will decrease the voltage threshold of a cell as a function of time. Another example comes from the capacitive effects occurring when neighboring cells get programmed, which increases the voltage threshold of every inactive cell nearby. Sometimes, this variation of voltage can be sufficient to shift towards another voltage level, resulting in a bit flipping. The probability for this event to occur is tightly coupled with the cell condition. A damaged cell will leak significantly more charges than a healthy one. Hence, the error rate allows us to appreciate the current condition of a cell. Yet, other factors might influence the instantaneous error rate and it is important to average them out to remove any bias in this type of measurement. For example, as illustrated in Figure 3.2, the error rate is very sensitive to the written data and will vary significantly from one cycle to another.

Counting the amount of faulty bits simply consists in counting how many bits are different between the written and read data. In a real system, counting the number of faulty bits is left to the ECC unit, which can evaluate from the coded data how many bits (or multi-bit

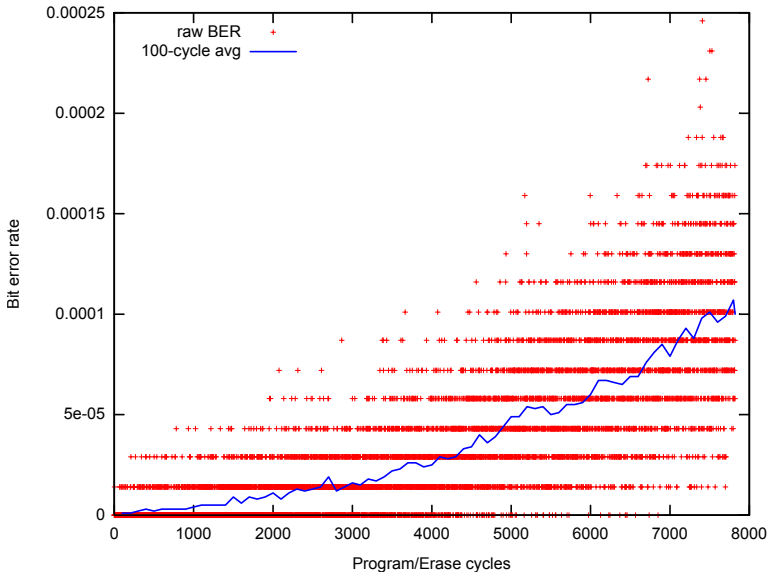


Figure 3.2: **Bit error rate with respect to P/E cycles.** The bit error rate evaluated right after programming a page is reported against the P/E cycles. The variance is significant from one measurement to the one that follows. Therefore, in practice, evaluating the current health of a page cannot be based on a single measurement. Instead, it would require to average a set of measurements.

symbols) are corrected. Of course, past a certain number of errors, the ECC unit capability would be exceeded and would prevent us to assess this number. Therefore, we do not rely on an ECC unit for our experiments; instead, we register the data that was written.

3.2.3 Energy

Another information that can be measured from outside the chip is the energy consumption. Thereby, the energy required to read, program and erase can be characterized. Further analysis would allow to model the energy consumption associated to the programmed data pattern. Yet, energy is one response that we did not consider in our evaluations.

3.3 Influencing Factors

The flash performance can be influenced by a large set of factors that we will list and describe in this section. In order to illustrate the way the main factors interact with the device, we propose a *gray-box* model of the flash page degradation in Figure 3.3 and will refer to it throughout this section. The inputs of this model include the data written to each page, the time between two accesses and the ambient temperature; for the output we only consider the BER here. Similar models could be built for access latency and energy consumption.

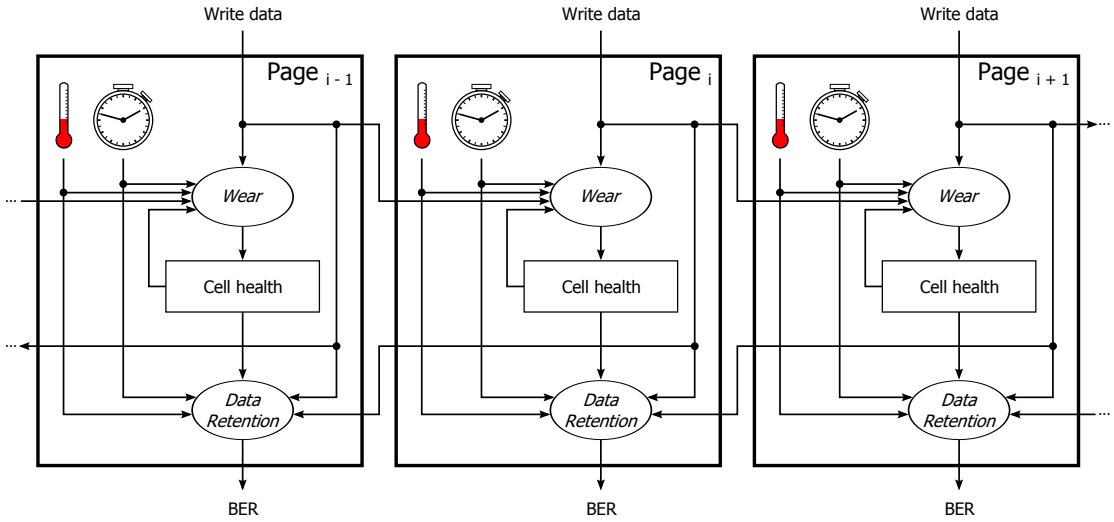


Figure 3.3: **Gray-box model of flash pages degradation.** This figures depicts a relatively high level and simplified description of the possible interactions between factors and flash page health. Several aspects are not considered here such as cells being in MLC or a cell-to-cell interference limited to direct neighbors.

3.3.1 Cell Condition

The cell condition drives the retention capability of the memory cells. Worn out memory cells will leak charges more rapidly than healthy cells, rendering them unreliable at some point. In Figure 3.3, this interaction is represented by the arrow between the *Cell health* and *Data retention* boxes. Furthermore, as discussed in Section 3.2.1, damaged cells will tend to get programmed faster than brand new cells [22, 17], which has been verified with our experimental setup. One side effect of a faster programming is that each programming steps become larger and therefore less precise; consequently, the probability to overshoot a cell programming increases. Furthermore, the cells' condition degrades with P/E cycling, hence it can also be considered a response or consequence of a characterization process. Yet, the degree of damage experienced by a cell cannot be probed directly from typical interfaces. Therefore, as discussed in Section 3.2.2, we estimate the cells condition of a page by relying on the fact that it correlates with the BER.

3.3.2 Write Data Pattern

The data to be stored on a cell is encoded into a voltage level. Hence, the written data pattern defines the voltage levels being programmed in a page, which is a factor that correlates with several effects that we will discuss in the next three subsections.

Wear

A first effect that correlates with the data patterns written to flash is the wear, or damage, associated to each programming level. This correlation is illustrated in Figure 3.3 by the path between the *Write data* input and the *Cell health*. The damage done to a cell depends largely on the actions that are applied to it. Accordingly, if a cell stays at the erased state during a P/E cycle, fewer charges will transit through the oxide layer; consequently, it will suffer less damage than if it were programmed to a higher level. This effect is often overlooked for a regular use case, where we can assume a uniform distribution of cell programming levels, which can then be averaged out. However, it becomes a key aspect for this thesis, for which we purposely unbalance the programming levels within blocks (see Chapters 4, 5 and 6). Accordingly, in this thesis, we will present a characterization experiment to analyze the degradation speed of cells for most of the technique that we will introduce.

Voltage Level Reliability

The second effect relates to the retention data capability: all voltage levels are not equal in this aspect. This effect on the retention is illustrated in Figure 3.3 by the arrow between the *Write data* input and the *Wear*. For example, the leaking current that empties progressively the memory cells correlates with the amount of charges trapped in the floating gates, therefore on the voltage level. Accordingly, the lowest voltage levels are more resistant to leakage, especially the erase state, which is the lowest level and will not change its state by losing charges. Reversely, the highest voltage level are less affected by capacitive effects than the lowest ones. Cai et al. give some insights on that topic by experimenting on a flash chip and reporting a bit error rate breakdown for every voltage level [6]. Overall, the data retention capacity of every voltage levels will largely depend on the threshold levels defined by the manufacturer. It will also be influenced by the longevity of the stored data, which defines how long cells will leak before being updated.

Neighboring Disturbance

The last effect concerns the disturbances that occur between neighboring cells. Flash memory technology is dense; consequently, the floating gates that capture the charges are relatively close to each other. This proximity incurs capacitive effects that have an influence on the storage reliability: when a cell is programmed to a higher voltage, the resulting voltage shift pushes the neighboring cell voltage a bit higher. This effect is illustrated in Figure 3.3 by the path between the *Write data* of Page i to the BER output of Page $i-1$. During a programming phase, the current neighboring capacitive effects are taken into account when charging the floating gates; hence, past interferences can be 'absorbed' that way. Indeed, the program/verify approach rely on the current voltage threshold that is read on a cell, and it will stop programming a cell, when an absolute voltage level is reached. Yet, after a cell is programmed, it is very likely that a neighboring cell will be programmed in the near future

(except for the last page of a block). This neighboring programming increases the capacitive effect in function of the voltage shift applied on the neighboring cells: a larger voltage shift will generate more disturbance. Hence, the worst-case disturbance for a cell is to have its neighbors programmed from the erase state to the highest voltage level, while the cell stays at the same level. In typical MLC devices, this scenario can only be experienced by cells staying at the erased state; in any other state, the partial programming makes this scenario impossible. This explains why manufacturers set a larger step between the erased state to the next voltage level than any other step. For SLC devices, there are two levels, so the worst case happens frequently, but does not have the same consequences than MLC devices, due to its large voltage threshold margins. Therefore, the data that is programmed on a page has a significant effect on the data that is stored on its neighbors. Accordingly, the experiment setup must take proper care to avoid writing patterns that would severely bias the results.

3.3.3 Time

A third factor affecting the storage reliability is time. As seen in Chapter 2, flash cells leak charges over time. The influence of time on data retention is pictured in Figure 3.3 by the arrow between the clock and the *Data retention* block.

Although time has a negative effect on the stored data, it also has a recovery effect on the cell health. This influence is illustrated in Figure 3.3 by the arrow between the clock and the *Wear* block. The stress sustained by flash cells during P/E cycles translates into charges being trapped in the cell oxide layer, which weaken its insulation property. Reversely, during long periods of resting time absent of any programming of erasing process on a set of cells, charges progressively get detrapped from the oxide layer, which restores somehow the oxide properties and the corresponding cells' health. This effect is known as the *recovery process* [45].

When characterizing the BER with respect to factors other than time, it is important to ensure that the various factors tested do not significantly change the experiment total time. For example, assuming an experiment based on two different benchmarks with one taking three weeks of cycling and the other half of it, their results would not be entirely comparable, as the second benchmark would not have benefited the same recovery process level.

On the other hand, if time is the desired factor to consider, experiments requiring to assess the flash cells state after long period of time will obviously be time consuming and not practical. Fortunately, in some cases, temperature can accelerate the effects being characterized and may reduce significantly the experiment time.

3.3.4 Temperature

Temperature is a factor that influences reaction rates by providing more or less energy to the particles of a system. Concerning flash memory, electrons at a higher temperature will have a higher probability to have the energy necessary to leave the floating gate; therefore,

high temperature increases the chance of charges to leak and reduces the retention time of cells. Similarly, temperature affects the charges trapped into the oxide layer and the recovery process described in the previous subsection. Accordingly, Lue et al. [40] suggest extending regular flash memory architectures by inserting local heaters, which would increase the temperature of blocks being erased to high levels in order to heal them in the process and, as a result, increase flash memory endurance. Furthermore, the temperature factor is used by manufacturers to emulate in a short time the charge loss that would occur on a long period. Thereby, it becomes possible to estimate the retention time corresponding to a given cell state and error correcting strength. Hence, in Figure 3.3, temperature interactions in the system are similar to time. In our experiments, we will also use this fact to verify whether the techniques that we propose are affected by time.

3.3.5 Reference Threshold Voltage

Besides the reliability difference between threshold voltage levels discussed in Section 3.3.2, the reference threshold voltage during the read out of a page can be a factor influencing the amount of faulty bits that we read. In some very recent NAND flash chips, manufacturers give access to advanced internal control of their decoding circuits. Specifically, it enables flash controllers to set the voltage threshold references that are used to read a page. Thereby, voltage shifts due to charge leakage over significant amount of time can be addressed to a certain extent by adapting the voltage threshold accordingly. This factor can be useful to assess the voltage shift over time or to characterize the variance in the cell voltage distribution after some specific manipulation. In our case, we did not have access to such flash memory chips, but its potential for characterization has already been demonstrated by Cai et al. [7]. We did not include this factor in the model of Figure 3.3.

3.3.6 Physical Cell Position

The cell and page position within a block affects its exposure to stress. The stress experienced is largely dependent on the events occurring in the neighborhood. Hence, cells and pages physically located on the boundaries of a block will generally show different degradation speed than pages located in the center. Furthermore, depending on the page programming sequence and page mapping architecture, the cell-to-cell interference might be unbalanced between one page to another. Hence, when characterizing mechanisms applied on a subset of pages within a block, it is important to make sure that the results are independent from the page position to not bias the results. Typically, alternating the pages from one block to another on which the studied mechanism is characterized will allow averaging out the physical position factor.

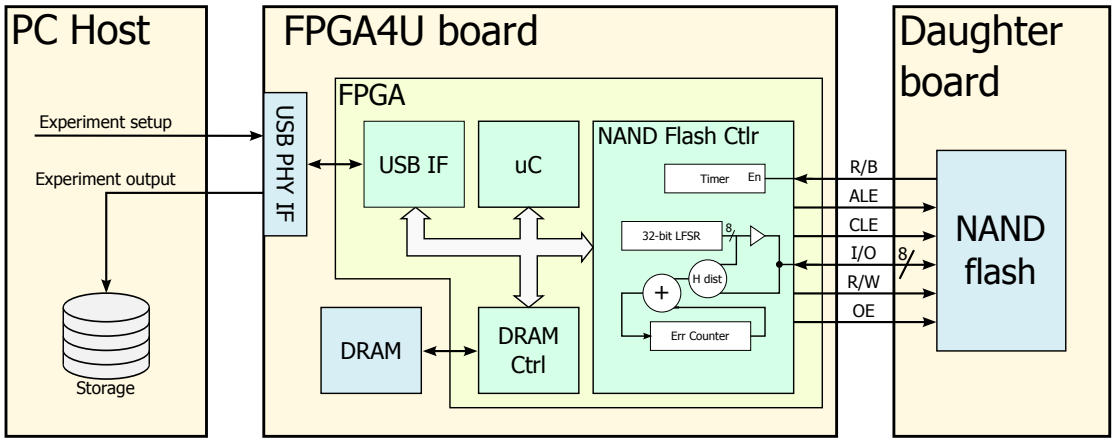


Figure 3.4: **NAND flash experimental setup system architecture.**

3.4 Experimental Setup

In this section, we present the experimental setup that we built for characterizing NAND flash chips and describe several specific features that we introduced in order to have full control over the factors presented in the previous section.

3.4.1 Architecture

The system is based on an FPGA board [19] that embeds a USB interface used here to collect the experiment output from a computer, and various *General Purpose I/O* (GPIO) pins, which are used in this setup to interface NAND flash chips. The general architecture is illustrated by Figure 3.4. We interface ONFI [49] compliant TSOP-48 NAND flash chips, which is a package commonly found in USB stick drives and SSDs. Those chips do not implement any address translation and let us access directly the raw flash storage, which is not possible with typical flash storage media such as SD memory cards or flash drives.

Characterizing flash memory often requires reading and writing thousands of GB of data in total. Therefore, it is vital to optimize the time it takes to perform a single read or write transaction. For this, we developed a custom flash controller that is optimized for our experiment process. Specifically, it features a 32-bit *Linear Feedback Shift Register* (LFSR) module to read-/write pseudo random pattern from/to the NAND flash memory, a register to count the number of faulty bits, and a timer connected to the Ready/Busy line of the NAND flash chip. The timer is used to characterize the time it takes for read, program and erase commands to complete. The error counts and timing data are transmitted through USB to a host computer, which can afterwards perform statistics on the experiment results. For a 1-week experiment, this data represents a total size on the order of 10 GB.

3.4.2 Characterization Procedure

Characterization experiments usually consists of repeating the followings steps thousands of times: (1) erasing every block considered for the experiment, (2) programming them with random data, and later (3) reading the data back to count how many bits flipped in the storage. Although a few variations are applied for specific experiments, this is the general procedure of an experiment.

The three steps described correspond to one experiment cycle. During our experiments, we make sure that the cycle frequency stays constant overall the experiment in order to have balanced and comparable cycles. Indeed, not only the flash timings change over time, but also the experiment might involve cycles with variable amount of data written. Hence, if not safeguarded, some cycles would be significantly shorter than others are and generate a bias on the flash characterization. Intuitively, one can think that this bias could come from variable retention times: shorter cycles would leave less time for the cells to leak after being programmed. However, although one cycle might last multiple times longer than another, a single cycle over 100 blocks generally takes less than five minutes to complete and over this time scale the retention effect is not significant. Yet, we can observe a bias from the recovery time, which applies over the total experiment duration. Thereby, for an experiment that takes weeks, the recovery time reduction starts to become significant with the P/E cycles progressively becoming shorter because of the programming latency getting shorter. This bias is not representative of a typical use case: an application will not necessarily program a device more often simply because the programming latency lowers. Therefore, we fix the cycle period to the slowest cycle for the complete experiment.

In order to prevent unbalanced written pattern to bias the results, we write pseudo-random data into the flash. This is performed with the dedicated LFSR, which avoids having to go through the CPU or the memory to read and write the flash memory and maximizes the throughput. Thereby, in order to program a flash page, the CPU only has to initialize the LFSR with a pattern that is kept in memory for later reference. Thereafter, the LFSR directly feeds the flash memory bus with consecutive bytes. To read back the random pattern and count the number of faulty bits, the CPU initializes the LFSR with the pattern that was used to write the page previously and initiate a read procedure. For each byte read, we accumulate into a register the hamming distance between the byte coming from the flash memory bus and the LFSR. At the end of the page reading, the register will report the total number of faulty bits.

While these safeguards are sufficient to provide reliable results in the general case, specific experiments might require more caution in the design of experiment. These will be discussed in the next chapters together with the corresponding experiments.

3.5 Related Work

Many researchers rely on characterization to propose different storage strategies, to define models describing low-level aspects of device, or simply to give insights on unpublished properties. We will list some of them here.

Mielke et al. [45] study the recovery process of flash memory cells discussed in Section 3.3.3. Over time, when the cells are not stressed, trapped charges tend to leave progressively the oxide layer, which corresponds to the cell healing or recovering. Accordingly, Mielke et al. let the cell recover for variable amount of time to characterize the recovery effect. In particular, they use the fact that high temperatures accelerate this effect to simulate long periods by baking the flash in an oven at 125˚C.

Joo et al. [32] designed an energy characterization platform for NOR flash memory chips to measure the energy consumption associated to each programming level and find significant differences between them. Accordingly, they propose to trade off some of the storage density for energy reduction by favoring some data patterns with an *energy-aware* encoding approach that minimizes the data programming energy cost.

On a similar idea, Grupp et al. [22] designed a characterization platform for NAND flash memory chips to measure the energy consumption and latency associated to the programming, reading and erasing. Without prior knowledge on the partial programming scheme of MLC described in Section 2.3, they identify the difference in energy and time required to program LSB and MSB pages. Furthermore, they acknowledge the fact that the programming time decreases with the cells aging. Accordingly, they propose several strategies to adapt the write performance depending on the workload.

Desnoyers [17] characterizes the performance and endurance of several SLC and MLC NAND flash chips and compares his measurements with the numbers specified by the manufacturers. Similarly to Grupp et al., Desnoyers finds that the programming latency decreases as the cells get weaker. Furthermore, he evaluates the degradation speed and endurance of the chips and estimates it to be around two orders of magnitude larger than specified. Yet, during this evaluation, only a single page is worn out, which does not allow assessing the disturbance effects. Furthermore, the written data is read right after being programmed; therefore, the retention time is not considered, which would be essential to compare the endurance with the specification.

Later, Grupp et al. [23] characterized a large set of SLC and MLC NAND flash chips from various node process sizes. They evidenced the degradation of NAND flash memory characteristics, when going for smaller process sizes and larger densities. Both latency and endurance are degraded with smaller cells and if this trend persists in the future, some important characteristics of flash memory storage will not be as appealing as they are today. Our work contributes to find original and architectural solutions to break this tendency.

On a 30 nm class NAND flash chip, Cai et al. [6] characterize separately various sources and types of error. Examples of error sources and types that they consider are read disturb, program disturb, erase errors and retention errors. Furthermore, they identify pages within a block that are systematically more reliable than their neighbors are. They compare how each type of error contributes to the global BER and conclude that the retention time is the most significant factor, although it is difficult to compare factors having completely unrelated units.

Other pieces of work [13, 4] characterize full flash storage systems, which include the FTL. In this type of experiments, the results are mainly dependent on the FTL efficiency rather than the underlying characteristics of the flash. In this thesis, we restrict our characterization to low-level physical properties and exclusively experiment on raw NAND flash chips.

All these pieces of work helped progressively the scientific community to get a deeper understanding of flash memory [18]. We hope that our work will serve the same purpose and help the community to propose new relevant storage strategies.

3.6 Conclusion

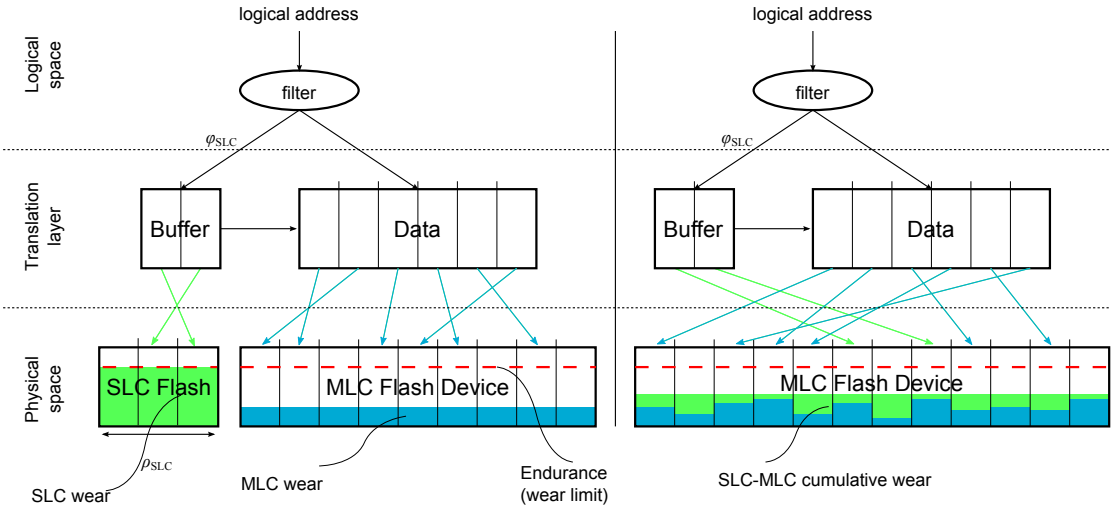
Experimenting on real flash memory is the most reliable way to validate assumptions when defining novel ways to access the flash memories that involve unspecified mechanisms. Designing reliable experiments highlighting only the effects from the set of factors that we are interested in requires understanding the various factors influencing flash performance. Many bright ideas have to be rejected simply because they are built on wrong assumptions. In this chapter, we detailed the various factors that must enter into consideration when characterizing neglected physical properties of NAND flash memory. We presented our experimental setup and described how to prevent undesired factors to bias the results. We discussed the general procedure of the experimentation, leaving more specific aspects to be described later, in corresponding chapters. In the next chapters, we will propose unconventional methods that change the way flash is degraded. For those methods, it is crucial to provide a good characterization of their effects in order to qualify them adequately for actual storage systems.

4 Libra: Balancing Mixed SLC-MLC Wear

In this chapter, we present and characterize a first example of neglected physical property. Specifically, we evaluate on real flash chips the wear difference when programming one versus two bits in an MLC. Thanks to these experiments, we contradict previous beliefs expecting that writing a single bit would reduce the total number of bit writable during a flash device lifetime compared to a regular use of MLC. With these findings, we bring flash storage devices more flexibility and potential to improve both performance and lifetime.

4.1 Introduction

As discussed in Section 2.4, Hybrid-FTLs [15, 16, 36, 37, 39] try to simultaneously achieve the benefits of coarse and fine grained mappings by dividing the flash memory into two regions: (1) a large data partition, which is addressed at the block-level, and (2) a small log buffer partition (typically less than 10% of the storage capacity), which is addressed at the page-level. Considering that a significant amount of write accesses gets directed to the small buffer partition, previous work [11, 26, 52, 47] proposed to build the small *buffer* partition from SLC flash, which provides high performance and low energy consumption but poor density, and the larger *data* partition on MLC of lower performance but higher density. As a result, the flash device has the potential to exhibit performances comparable to SLC (particularly for frequent local updates) while keeping the area efficiency of MLC to a great extent. However, these authors largely disregarded the effect of such SLC-MLC partitioning on the device lifetime. All the previous pieces of work suggest managing the SLC and MLC partitions as distinct physical parts, which can lead to a serious reduction in lifetime. We show that such a configuration can reduce the lifetime by more than half compared to a regular MLC device, assuming typical buffer sizes and utilization. Importantly, MLC endurance is already one order of magnitudes shorter than SLC endurance [23]. Consequently, any further reduction of lifetime may jeopardize the use of SLC-MLC partitions in a practical system, despite their significant advantages in performance and density. Figure 4.1(a) suggests how the extensive use of the buffer partition, due to a particular application write pattern, results in an unbalanced



(a) Hard Partitions (b) Soft Partitions

Figure 4.1: **Hard versus soft partitioning.** A hybrid-FTL redirects small writes to a page-level mapped buffer and directs large sequential writes directly to the block-level mapped data partition. The buffer uses SLC to benefit from low write latency and low energy, while the data uses MLC for density. When writes are unbalanced across buffer and data, a hard partition will wear faster than the other, while soft partitioning (the contribution of the present chapter) allows balancing the wear on the global device.

stress causing the device to fail well before most of its cells deteriorate above their maximum wear level (the large data partition is still healthy).

Accordingly, we introduce in this chapter *Libra*, a soft SLC-MLC partitioning architecture that maximizes the device lifetime by dynamically changing the physical allocation of the buffer in order to balance the cumulated stress of each individual flash block. Such technique relies on the fact that an MLC can be managed as an SLC while largely keeping the performance benefits of a physical SLC. Figure 4.1(b) illustrates a device implementing *Libra*, where the buffer uses SLC-mode and each cell has a cumulated wear from MLC- and SLC-mode that can be globally balanced. The proposed soft partitioning is built from a single flash technology instead of two for the hard partitions, which simplifies many aspects of the storage architecture. Furthermore, it can be adapted to existing hybrid-FTLs with minimal effort to significantly increase the device lifetime (between 1.5–10× for typical scenarios), while displaying the same benefits in performance, energy and density than hard partitioning. *Libra* is practical and attractive, enhancing an MLC device with performances close to SLC at a modest penalty in density while still being able to provide lifetimes slightly superior to MLC at virtually no extra cost.

4.2 SLC-MLC Hybrid Storage

In this section, we introduce SLC-MLC hybrid storages and provide a model to quantify their lifetime. MLC devices store multiple bits per memory cell providing a larger bit density and hence a smaller cost per bit. However, manipulating MLCs is trickier than SLCs: the higher precision required to differentiate the multiple voltage levels translates into about 3–4× slower page programs and consumes more energy [22]. Furthermore, because of reduced margins between the voltage thresholds, MLC is more sensitive than SLC to charge losses and neighboring cell interferences that typically affect flash reliability, which translates into about an order of magnitude shorter endurance [23]. Therefore, MLC offers a higher bit density than SLC at the expense of a lower performance, higher energy consumption, and reduced lifetime.

Hybrid flash devices combine one or more SLC devices to act as buffer with one or more MLC devices to implement the data partition; their purpose is to improve the device performance: the more hot data (frequently updated data) directed to the log buffer, the closer the hybrid device performance is to that of an SLC-only device. Log buffers need to be carefully dimensioned and the smaller the buffer partition can be made, the higher the bit density of the flash device. This is a well understood trade-off between cost and performance [52]. Yet, the impact of such partitioning on the device lifetime is critical and must be carefully considered.

Depending on the application write pattern, an unbalanced wear can occur between the buffer and data partitions, as illustrated in Figure 4.1. Each partition lifetime is proportional to its technology endurance and capacity, and inversely proportional to the ratio of writes directed to it. For example, let us consider a budget of 100 cells, allocate 5% for an SLC buffer and the rest to the MLC data partition. Considering that the endurance of an SLC is about 10 times larger than of an MLC [23] and, for this particular example, that each partition receives 50% of the writes, in this scenario, compared to an MLC-only device receiving 100% of the writes, the normalized MLC-data partition lifetime is

$$L_{\mathrm{D}} = \frac{\frac{Capacity_{\mathrm{DATA}}}{Capacity_{\mathrm{TOTAL}}}}{\frac{Writes_{\mathrm{DATA}}}{Writes_{\mathrm{TOTAL}}}} = \frac{0.95}{0.5} = 1.9. \tag{4.1}$$

On the other hand, the SLC partition allocates only 5% of the cells and each cell has ten times the endurance of an MLC but can store only half of the bits of an MLC, which translates to 2.5% of the capacity of an MLC-only device; accordingly, the normalized lifetime of the SLC partition corresponds to

$$L_{\mathrm{B}} = \frac{\frac{Capacity_{\mathrm{BUFFER}}}{Capacity_{\mathrm{TOTAL}}} \cdot \frac{Endurance_{\mathrm{SLC}}}{Endurance_{\mathrm{MLC}}}}{\frac{Writes_{\mathrm{BUFFER}}}{Writes_{\mathrm{TOTAL}}}} = \frac{0.025 \cdot 10}{0.5} = 0.5. \tag{4.2}$$

This indicates that a device with such a hybrid configuration will last half of the time of an MLC-only device, which is already significantly shorter than the lifetime of an SLC-only device.

In order to model analytically the lifetime of a hybrid flash device, we define ϕ_{SLC} and ϕ_{MLC} as the proportion of writes directed to the buffer and data partitions, respectively, with $\phi_{SLC} + \phi_{MLC} = 1$. Let ρ_{SLC} and ρ_{MLC} respectively be the ratios of the device's cells allocated to the buffer and data. We define L_B and L_D as the buffer and data partition lifetimes, functions of the partition size and ratio of writes directed to it. The partition lifetimes are normalized to an MLC-only device's lifetime that would receive 100% of the writes. We will use this MLC-only baseline as a lifetime reference throughout this chapter. Considering an n-bit per cell technology and an SLC endurance comparatively γ times larger, the lifetime of the buffer L_B is

$$L_B = \frac{\gamma \cdot \rho_{SLC}}{n \cdot \phi_{SLC}}. \tag{4.3}$$

The data partition lifetime is expressed as follow:

$$L_D = \frac{\rho_{MLC}}{\phi_{MLC}}. \tag{4.4}$$

A device on hard partitions will die as soon as the first of its partition wears out. Accordingly, a hard partition lifetime corresponds to the minimum out of its partition lifetime:

$$L_H = min(L_B, L_D). \tag{4.5}$$

Assuming MLC (n=2 and γ=10), Figure 4.2 plots Equation (4.5) and represents the device lifetime, normalized to an MLC-only device, for different buffer sizes ρ_{SLC}, and function of ϕ_{SLC}, the ratio of writes directed to the log buffer. We observe that for reasonable buffer sizes (i.e., ρ_{SLC}<10%), the lifetime of hybrid devices drops significantly when more than 25% of writes are directed to the buffer. Around one fifth of the cells should be allocated to the buffer to ensure a lifetime close to the MLC-only's. Because the buffer does not account for capacity, this would result in a significant density cost.

The main issue of hard partitioning is the inability to share the wear between its partitions. This, as shown in Figure 4.2, can seriously compromise the viability of hybrid devices. Therefore, part of the previous work on hard partitions proposed heuristics to restrict the hot write ratio artificially to a predefined range. However, this approach breaks the purpose of the buffer partition and will inevitably generate more garbage collection overhead and degrade performances. In the following section, we introduce *Libra*, which builds on soft partitions to share and balance the stress on the whole device and maximize its lifetime without being concerned by the hot write ratio.

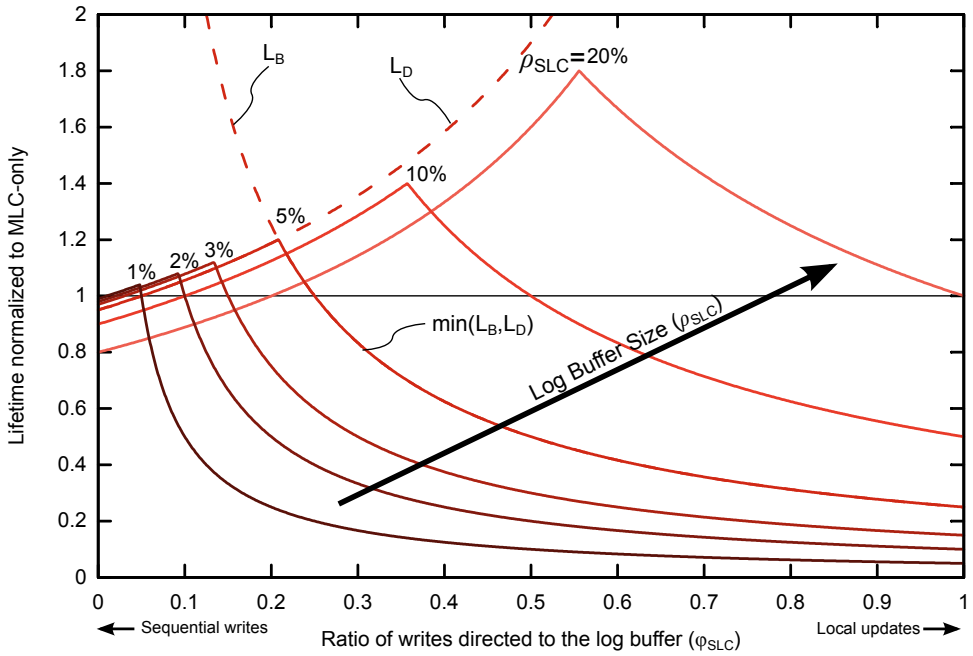


Figure 4.2: **Hard partitions lifetime model for different buffer sizes as a function of the write ratio to the buffer.** The model is normalized to a 2-bit MLC-only flash device lifetime. We illustrate the 5% buffer size lifetime construct by plotting both the buffer and data partition lifetime components. For large sequential writes, where a FTL will more likely bypass the buffer, the device lifetime is bounded to the data partition on the left. Small and frequently updated writes will wear out the buffer first, limiting the device lifetime to the buffer partition. For reasonably sized SLC buffers, lifetime is reduced by up to one order of magnitude compared to an MLC-only device.

4.3 Libra: Soft Partitions to Balance Wear

Libra relies on soft partitions to break the rigidity of hard partitioning by changing the physical placement of the SLC-mode log buffer depending on the device wear [28]. This is made possible by the fact that MLC can be managed in software as SLC achieving better performance. We have actually used real chips and validated experimentally that the performance of an MLC managed as an SLC are very similar to the ones exhibited by an SLC device. We propose the FTL to keep track of the cumulative wear (SLC- and MLC-mode) to decide dynamically the best physical allocation.

4.3.1 Faster MLC: Managing MLC as SLC

MLC can also be used to store a single bit instead of two and recover the performance and energy consumption benefits of SLC [54, 22]. Figure 4.3 illustrates the programming sequence of a 2-bit MLC, as described in Section 2.3 and that we remind here. Interestingly, programming only the LSB of MLC shows performances very similar to SLC, which motivated previous

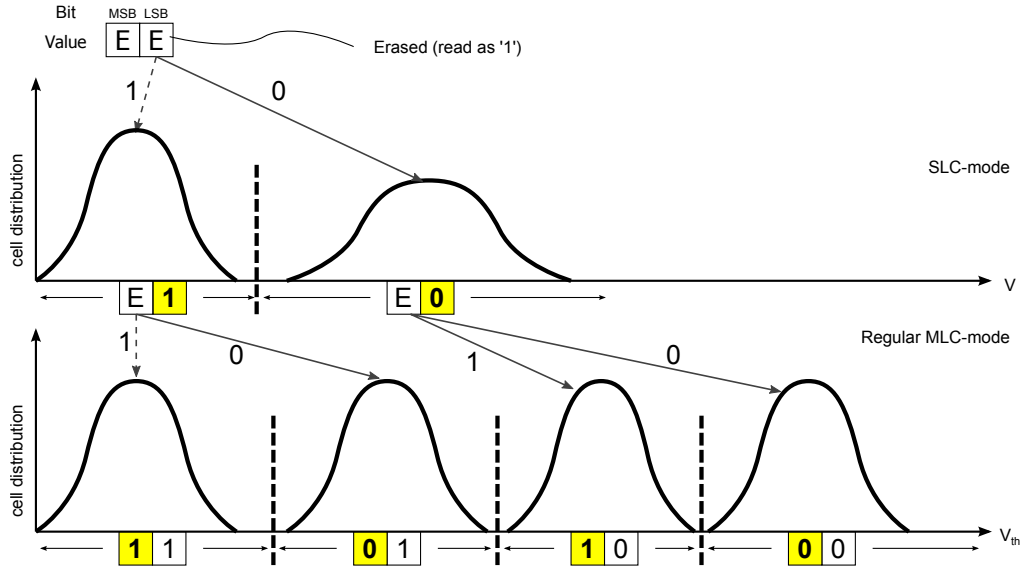
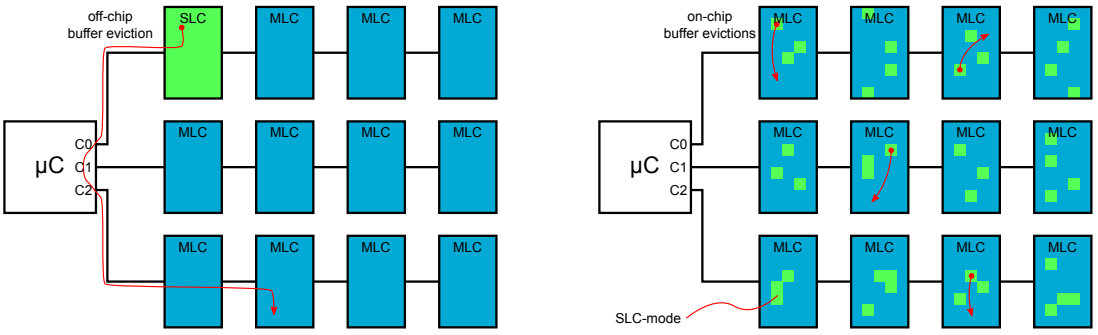


Figure 4.3: **Programming of a 2-bit MLC.** Using cells in SLC-mode consists of programming only the first bit of each cell.

researchers [38, 22] to propose the use of MLC as SLC to benefit from higher performances opportunistically. This is done without breaking the page programming sequentiality constraint nor compromising the stored data. Thereby, performance is obtained at the expense of density in an MLC device. Such way of manipulating an MLC is referred to as *SLC-mode* in this thesis. In this work, we propose to go one step further and dynamically change the physical allocation of the buffer to globally balance the wear. Accordingly, when a block allocated to the buffer has accumulated significantly more wear than a data block, both blocks will be swapped.

4.3.2 Software-Controlled Log Buffer

Whereas hard partitioning is typically built on heterogeneous SLC and MLC hardware, the soft-partition scheme applies to a completely homogeneous hardware architecture made only of one or more MLC chips. Figure 4.4 illustrates soft and hard partitioning examples for a possible architecture made of a microcontroller connected to three channels of several flash chips. On Figure 4.4(a), the device built on hard partitions has multiple MLC chips and a single SLC chip, whereas on Figure 4.4(b), the device is composed exclusively of MLC chips using soft partitions. Architecturally, soft partitioning can offer many benefits. In this example, as opposed to hard partitions, the bandwidth to the buffer would not be limited to one channel. Instead, because the buffer can be distributed on all the chips, multiple channels can be accessed in parallel to write the buffer, as well as multiple chips in an interleaved fashion. Furthermore, the evictions from the buffer do not necessarily require an expensive off-chip migration, but can potentially be performed on-chip. Finally, soft parti-



(a) Hard Partitions (b) Soft Partitions

Figure 4.4: **Hard and soft partitioning architecture examples.** Hard partitioning relies on a mix of SLC and MLC devices, while soft partitioning is built on a homogeneous MLC fabric. On both example, the storage controller is connected to three channels with four flash chips each. On hard partitions, only one chip is SLC, which is the only one allocated to the buffer. While on soft partitioning, the buffer can be distributed on every chip. This enables multiple advantageous features that are not covered here, such as cheap on-chip buffer-to-data migrations, large write bandwidth to the buffer, or a resizable buffer.

tioning does not restrict the buffer size to a physical constraint; hence, in order to improve performance, the buffer size could be expanded when the device capacity is not completely used or by using MLC-mode for some of the buffer blocks, dynamically trading off write latency for buffer capacity. Yet, in this work, we decided to not cover those advantages and focus on the raw benefits of soft partitioning. Thereby, the changes to implement *Libra* are kept minimal, while its improvements are already significant and conservative. *Libra* is able to write selectively regions of the flash chip(s) to SLC-mode or MLC-mode at will, with the intention to distribute the wear evenly throughout the whole device. While small buffers are likely to die first for hard partitions, soft partitioning can spread the localized stress over the complete device.

Classical wear leveling algorithms periodically switch cold and hot blocks in order to even their P/E cycle counts and balance the wear on the whole device. Typically, when a hot block is evicted from the log buffer and erased, the wear-leveling logic compares P/E counts of this block with the coldest block (i.e., with the smallest P/E count) and decides whether to swap them; for example, when the P/E count difference reaches some threshold. Upon a swap, every page of the cold block is copied into the evicted hot block, and the cold block is then erased and allocated to the buffer. Figure 4.5 illustrates the evolution of a device mixing SLC- and MLC-mode on soft partitions and shows how careful wear balancing can avoid the premature death that is likely to happen in the hard partitioned architectures proposed in all previous work. Focusing on the leftmost physical block, one can see how this is initially allocated to the buffer, thus managed as SLC, then is invalidated and freed from both partitions to be later allocated to the data partition, managed as MLC. Such transitions are naturally

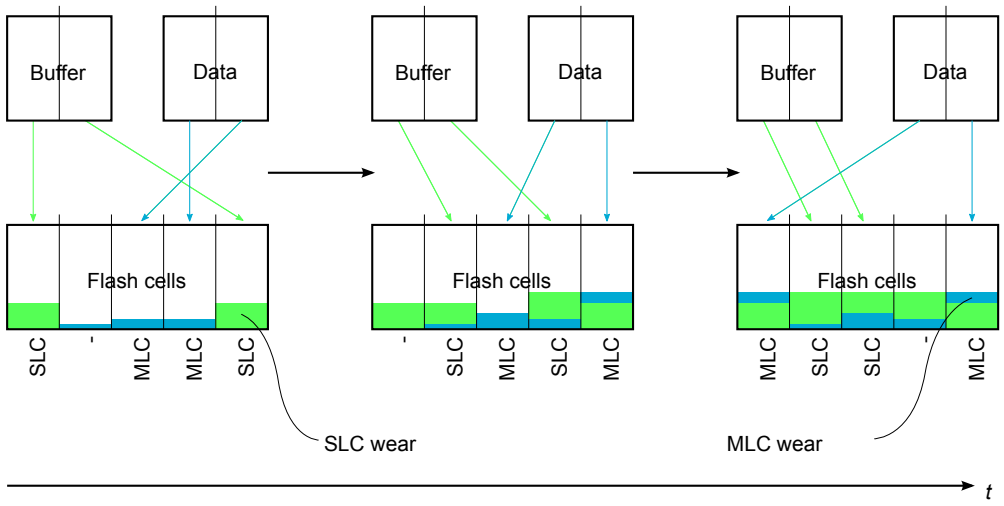


Figure 4.5: **Software-controlled log buffer.** A practical scenario of a hybrid-FTL, where blocks regularly alternate between SLC- and MLC-mode in order to balance the overall wear.

triggered by the wear-leveling algorithm. Notice that the wear of the block increases through time and is a result of both the periods when the block is programmed as SLC and as MLC.

4.3.3 Libra Implementation

Implementing *Libra* in hybrid-FTLs is straightforward, because hybrid-FTLs already incorporate the data structures and mechanisms required for it. Regarding the mode identification (i.e., SLC or MLC), hybrid-FTLs use one bit in the spare bytes of the first page of a block to differentiate a log-buffer block from a data block. Considering that every log-buffer block is used in SLC-mode with *Libra*, then the SLC-MLC classification is implicit to this differentiation. The only difference with a hybrid-FTL using exclusively MLC-mode, is that only half of the raw log-buffer capacity will be used. Accordingly, whenever we access a log-buffer block, only LSB pages will be referenced. This is illustrated in Figure 4.6, where block A has been set to SLC-mode. Importantly, it is not necessary to allocate memory in the translation table to save explicitly all MSB pages' state; indeed, when a block is used in SLC-mode, every MSB page is implicitly considered idle.

Although implementing *Libra* on hybrid-FTL is simple, it does not mean that it is bounded to this class of FTLs. In fact, it can perfectly be implemented by any type of FTL, such as page-level mapped FTLs. The only requirement is to be able to differentiate blocks allocated in SLC- and MLC-mode and have a level of decision choosing whether a data should be written in either of these modes. Generally, page-level mapped FTLs implement a hot/cold filter, much like hybrid FTLs, to regroup data likely to be updated soon together in the same blocks, resulting in hot blocks made of many invalidated pages, which reduces garbage collection overhead. Accordingly, for those FTLs, data categorized as hot could be written in SLC-mode. However, as opposed to classical hybrid-FTLs and hard partitions, the number of hot blocks

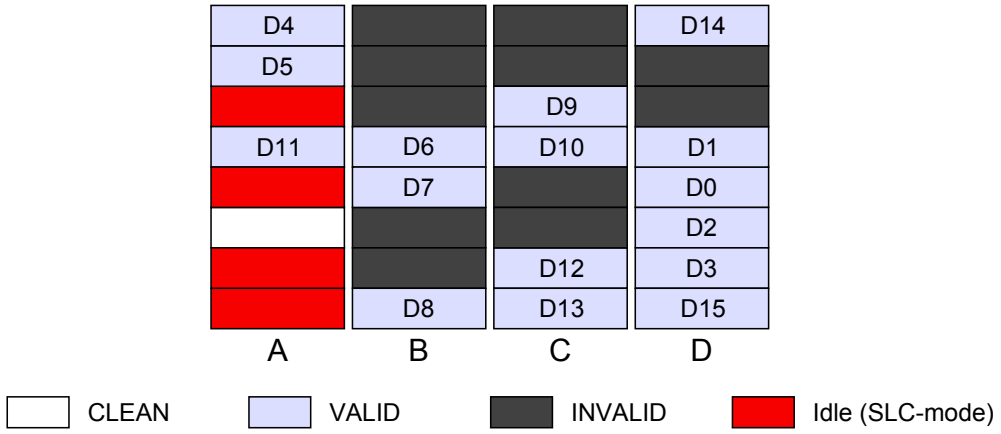


Figure 4.6: **Idle MSB page state.**

would not be bounded to a fixed number. Such flexibility can be exploited easily in page-level FTLs implementing *Libra* and would reduce garbage collection overheads compared to hard partitions schemes. However, it is difficult to evaluate the performance of page-level mappings accurately, because to start observing the effects of the garbage collection would require executing each traces many times, which would then bias the locality and the write ratio to the buffer. Therefore, we evaluate only hybrid-FTL architectures for a fair comparison, as their performance do not get improved by the use of *Libra* alone and the garbage collection effect can be observed much sooner. Thereby, we can concentrate on the lifetime improvements of our approach.

Importantly, the FTL needs to consider a global wear metric that includes the effects of both, the MLC-mode and SLC-mode, to be able to implement a regular wear leveling algorithm. Such metric is the foundation of our proposed *Libra* and will be detailed in the following subsection.

4.3.4 Libra Lifetime Model

We evaluate the lifetime of a flash device by the total amount of data written before wearing out. This lifetime is inversely proportional to the average wear experienced when writing a bit in a cell. Interestingly, this wear is correlated to the amount of charges being stored in a cell. Hence, a partial programming (i.e., SLC-mode) would generate less wear than a full programming (i.e., MLC). This is verified experimentally on real chips in Section 4.4. Let ω_{SLC} be the relative wear associated to writing a bit in SLC-mode with respect to the wear per written bit in MLC. *Libra* uses blocks alternately in SLC- or MLC-mode, while still being able to evaluate the cumulative wear of each individual block. Hence, the lifetime of a device implementing *Libra* is function to the write ratio directed to the buffer, ϕ_{SLC}. In the one extreme, when the MLC-mode is exclusively used (ϕ_{SLC}=0), the device lifetime is trivially equal to an MLC-only device. In the other extreme, when the SLC-mode is exclusively used (ϕ_{SLC}=1), the device

Table 4.1: 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	130 μs	40–60 μs
LSB write lat.	330 μs	360 μs
MSB write lat.	1,750 μs	1,800 μs
Erase latency	4 ms	3 ms
Architecture	ABL	HBL

lifetime is defined by the wear of the SLC writes and corresponds to $\frac{1}{\omega_{SLC}}$. Note that despite the larger read margin of the SLC-mode (larger than MLC, as discussed in Section 4.3.1), the lifetime is different from a regular SLC device, because every block can potentially later be allocated to MLC and is therefore restrained to the MLC endurance. Furthermore, contrarily to hard partitions, the lifetime with *Libra* is not directly dependent to the buffer size: regardless of the soft partition being written, any physical flash block can be addressed on the flash device. Still, similarly to hard partitions, for a given benchmark, the buffer size will influence the garbage collection overhead and ϕ_{SLC}. Given the write ratio directed to the buffer, ϕ_{SLC} the average wear normalized to the MLC wear is

$$1 - \phi_{SLC} + \omega_{SLC} \cdot \phi_{SLC} = 1 - \phi_{SLC} \cdot (1 - \omega_{SLC}). \tag{4.6}$$

Trivially, the lifetime of *Libra*, L_S, normalized to MLC-only is the inverse of Equation (4.6), which is

$$L_S(\phi_{SLC}) = \frac{1}{1 - \phi_{SLC} \cdot (1 - \omega_{SLC})}. \tag{4.7}$$

When $\omega_{SLC} \leq 1$, *Libra* ensures a lifetime larger or equal to MLC-only. However, ω_{SLC} is a parameter that cannot be found in typical specifications of MLC flash chips, as manufacturers generally do not publish the SLC-mode characteristics in their documentation. In the next section, we will describe how to extract this parameter experimentally from actual flash chips.

4.4 SLC-mode Characterization

In order to evaluate the relative wear of SLC-mode ω_{SLC} with respect to MLC, we use the FPGA-based platform described in Section 3.4 to extract experimentally this parameter for two 30 nm class NAND flash chips from different manufacturers, whose characteristics are listed in Table 4.1.

The experiment consists in programming continuously a set of fifty flash blocks either in SLC- or MLC-mode, while periodically measuring the BER. Specifically, we write random data in every block, read them back to identify and count fault bits, erase the blocks, and start over. We characterize several rates of SLC-mode by setting for each block a fixed predefined SLC-mode frequency ranging from 0% to 99%. We report the results on Figure 4.7 for five different SLC-mode frequencies and for the two chips. The graphs show how the BER evolves with respect to the P/E cycles for the different sets of blocks. The BERs are averaged over periods of 100 P/E cycles and, in order to measure the relative wear of the cells with the same reference for every block, only MLC cycles are considered. Indeed, the SLC-mode superior reliability systematically generates fewer bit errors than MLC. From the figure, we distinctly see an effect from the SLC-mode frequency on the degradation speed: the more frequently the SLC-mode is used, the slower becomes the degradation.

In order to quantify the effects of the SLC-mode on the device wear, we fit our results on flash degradation models. In previous work related to flash memory characterization [46], the growing BER in function of P/E cycles is generally modeled by the power function

$$BER(x) = \alpha x^{\beta} + C. \tag{4.8}$$

While this function fits relatively well the BER of individual pages, in our case, it did not fit that well when averaging the BER of a set of pages. For the studied flash chips, we found that adding a term of degree one to the power function fits accurately the average BER and becomes:

$$BER(x) = \alpha x^{\beta} + \delta' x + C. \tag{4.9}$$

We observed that the effects of SLC-mode will stretch linearly the reference curve on the P/E cycle axis. Accordingly, we adapt Equation (4.9) to propose the equivalent form

$$BER(x) = (Ax)^{\beta} + \delta Ax + C, \tag{4.10}$$

where β, δ, and C are constant for a chip. The A coefficient represents the degradation speed and varies in function of the SLC-mode frequency; this is the effect that we want to evaluate. We fit every set of data to Equation (4.10) by fixing the constant factors to the most satisfying values, which results in marginal sum of squared residuals. The fitted curves and their corresponding parameters are provided in Figure 4.7. Our experiment confirms that SLC-mode cycles generate less stress than regular MLC cycles and allows us to quantify it. *Libra* will use this information to evaluate blocks wear based on their SLC and MLC cycle counts. Yet, this experiment aggressively wears out the blocks by continuously writing them, which is not representative of a realistic usage of the device. Therefore, we propose in the next subsection another experiment that validates our measurements for a more realistic usage.

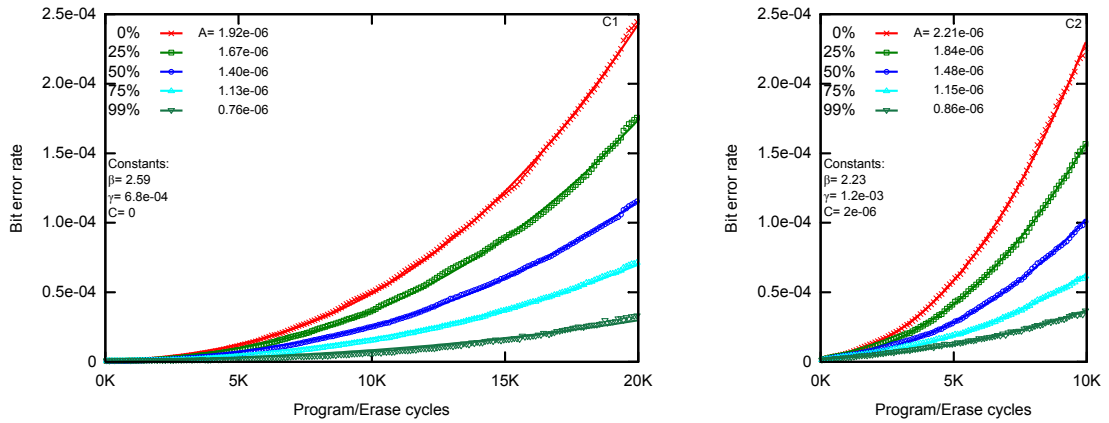


Figure 4.7: **Comparison of SLC- and MLC-mode cell-degradation speed.** P/E cycles are applied on five sets of blocks, each with a different SLC-mode cycle frequency. The BER is evaluated exclusively during MLC cycles. C2 degrades significantly faster than C1. For each chip, we report the fitted constant parameters and the variable parameter A, which varies with the SLC-mode ratio. As anticipated, SLC-mode cycles generate clearly less stress to the cells than MLC cycles.

4.4.1 Considering the Recovery Factor

Although applying continuously P/E cycles on flash cells allows reducing the experiment time, it does not represent a realistic scenario. In a real system, the cells are written at a much lower frequency, which gives time for trapped charges to leave the oxide after a while, healing the cells. This phenomenon is known as the *recovery process* [45, 46].

In order to take into account the recovery effect when evaluating the SLC-mode wear, we conducted a second experiment, similar to the first one, except that it includes periodic baking times. Every few thousands P/E cycles, we paused the experiment, removed the daughter board with the flash chips from the FPGA board, and baked them at 125°C for five hours. Baking the chips allows accelerating the recovery process significantly [45]. From now on, we will refer to the previous experiment as the *fast* experiment and the presently described experiment as the *baked* one.

We report our measurements on Figure 4.8. The recovery effect is significant for chip C2, whereas for C1, the effect is noticeable only after 15,000 P/E cycles. The reference curve is taken from the *fast* experiment, without SLC-mode cycles nor baking periods. For C2, the BER drops correspond to the baking events (marked by vertical bars), while the rapid BER growth corresponds to the aggressive P/E cycling. For clarity, the full C2 baked data is only provided for the set without SLC-mode cycles (0%). We only consider the BER when cells are freshly baked to fit the data on the model of Equation (4.10). Similarly to the *fast* experiment, we can observe that SLC-mode cycles infer less stress than regular MLC-mode cycles. Based on those measurements, we will precisely quantify and discuss the resulting SLC-mode wear in the next subsection.

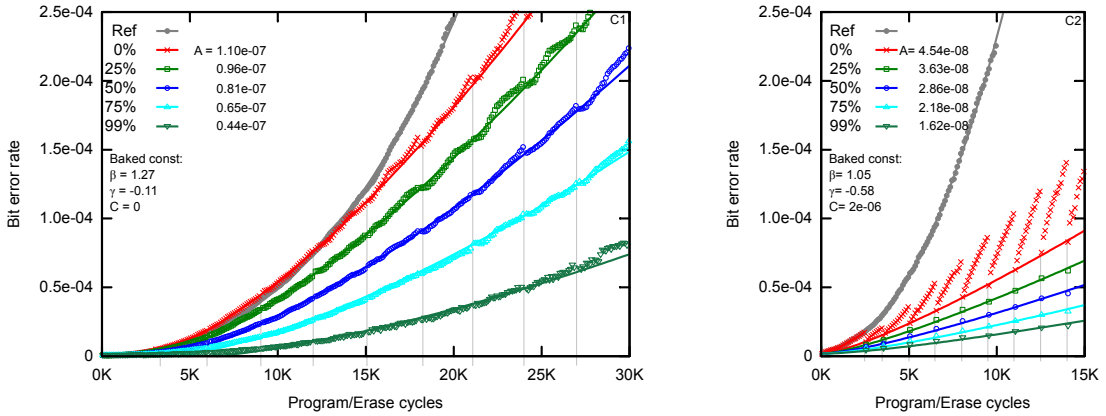


Figure 4.8: **Considering recovery effects.** We evaluate the combined effects of recovery and SLC-mode cycling on cells. We accelerate the recovery by baking periodically the chips. The baking events are marked by the vertical bars. The recovery effect is significant for C2, whereas for C1 it becomes observable only after 15,000 cycles. The reference curve corresponds to normal cycling without baking episodes. For visibility, the full data from baked C2 blocks is only reported for the 0% SLC-mode set. Only the data points measured right after the baking are fitted to the model to extract the blocks degradation speed. The relative wear of SLC-mode remains stable with the recovery process; it even gets slightly lower for C2, which means that SLC-mode is even less harmful when considering recovery time.

4.4.2 SLC-mode Wear

From our two experiments, we observed that the ratio of SLC-mode cycles has an effect on the degradation speed, which we quantified with the fitted parameter A. This parameter is directly proportional to the average wear of a P/E cycle. Hence, the smaller is A, the less the damage on a cell and the larger the endurance. For example, looking at the fitted parameters of C2, a 75% SLC-mode ratio (A_{75}=1.15·10^{-6}) almost halves the wear compared to a 0% SLC-mode ratio (A_0=2.21·10^{-6}). Accordingly, blocks with a 75% SLC-mode ratio will require about 2× the P/E cycles to reach the same BER than a block with a 0% SLC-mode ratio. In order to evaluate ω'_{SLC}, the relative wear of an SLC-mode P/E cycle compared to MLC, we express the average wear in function of ϕ'_{SLC}, the ratio of SLC-mode cycle, as

$$AvgWear(\phi'_{\text{SLC}}) = 1 - \phi'_{\text{SLC}} + \phi'_{\text{SLC}} \cdot \omega'_{\text{SLC}} \tag{4.11}$$

with $1 - \phi'_{\text{SLC}}$ corresponding to the MLC wear contribution and $\phi'_{\text{SLC}} \cdot \omega'_{\text{SLC}}$ to the SLC-mode contribution. For each chip, we fit Equation (4.11) on the extracted A parameters with a linear regression to extract ω'_{SLC}. Accordingly, we plot the fitted A parameters and their corresponding fitted curve in Figure 4.9 and also report the corresponding ω'_{SLC}. The data is normalized to the corresponding 0% SLC-mode ratio. The standard deviation associated to every data point is very small (less than 2%). The difference between the *baked* and *fast* results for C1 is not observable; hence, the C1 *baked* results have been omitted.

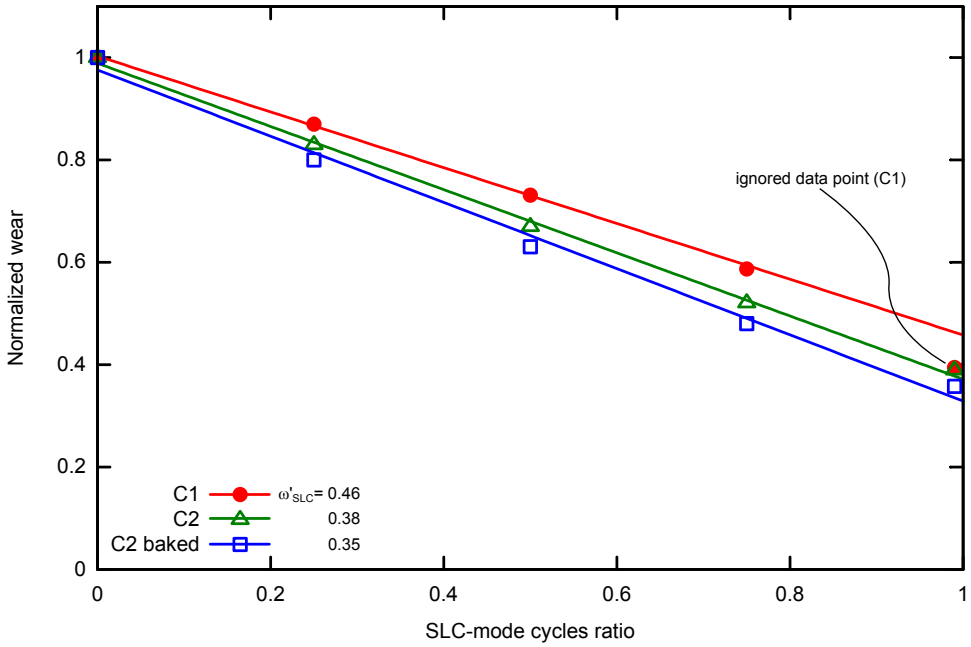


Figure 4.9: **Evaluation of SLC-mode wear.** We plot the average wear in function of the SLC-mode P/E cycles ratio as measured on our two chips and provide the corresponding SLC-mode P/E cycle wear coefficient (ω'_{SLC}) for each data set. For the C1 chip, the *fast* and *baked* experiments gave identical results. Thereby, the *baked* experiment data is only shown for C2. We observe for C1 an irregularity when measuring the endurance at 99% SLC-mode cycle ratio. Specifically, for this chip, the SLC-mode wear is reduced when approaching extreme SLC-mode ratios, while the wear factor stays constant on lower ratios. This effect is not observed on C2, where the SLC-mode wear coefficient stays constant. Hence, in order to stick with our simple model, we ignore this irregular (yet favorable) data point and evaluate for C1 a pessimistic SLC-mode wear that is accurate for the majority of SLC-mode ratios.

We observe a slightly smaller SLC-mode wear when we let C2 recover, while for C1 the difference is negligible. In the case of C1, we also notice that the SLC-mode wear is not constant: the data point at 99% of SLC-mode cycle ratio is not aligned with the previous ones. Specifically, the SLC-mode wear coefficient decreases when approaching to extreme SLC-mode ratios. Conservatively, although this fact would benefit the lifetime of *Libra*, we prefer to ignore this particular data point when evaluating ω'_{SLC}, in order to provide a more accurate SLC-mode wear information on the lowest SLC-mode ratios.

Having measured ω'_{SLC}, an FTL is now able to convert a mixed SLC- and MLC-mode wear into a global wear expressed in MLC P/E cycles. While, typical FTLs keep a single P/E counter for each block in order to balance the wear; *Libra* needs two counters in order to differentiate SLC- and MLC-mode P/E counts, which represents a negligible overhead of about 16 bits per block. Accordingly, *Libra* expresses the global wear of a block B with

$$Wear(B) = Count_{\text{MLC}}(B) + \omega'_{\text{SLC}} \cdot Count_{\text{SLC}}(B). \tag{4.12}$$

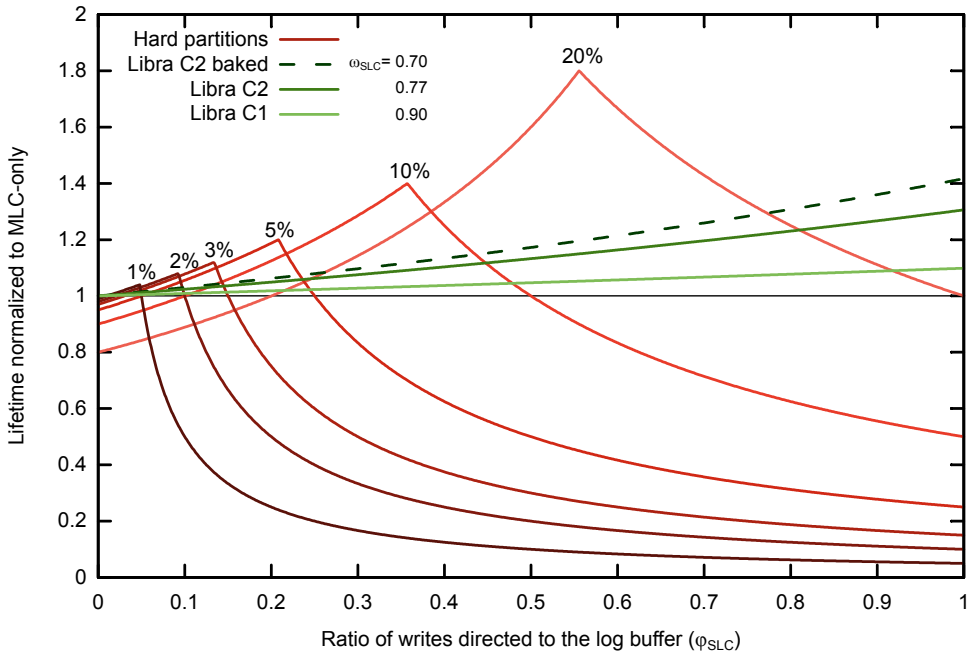


Figure 4.10: ***Libra* versus hard partition lifetime.** We provide the expected *Libra* lifetime corresponding to our chips characterization and compare them to hard partitioning. The lifetime is normalized to an MLC-only device. As opposed to hard partitions, the *Libra* lifetime is not function of the buffer size but solely of the ratio of writes to the SLC buffer. We provide for each *Libra* lifetime curve the corresponding SLC-mode wear coefficient ω_{SLC}. The lifetime evaluation corresponding to the *baked* experiment of C2 is dashed and indicates that the recovery effect observed in typical flash usage increases the lifetime extension provided by *Libra*. Accordingly, the lifetime evaluated from a *fast* experiment can be considered as a conservative estimation.

Thereby, when typical FTLs perform their wear-leveling based on the MLC P/E count of every block, *Libra* uses the result of Equation (4.12) instead.

Regarding the lifetime that can be expected by *Libra* for each chip, we rely on the lifetime model of Equation (4.7). First, we need to convert ω'_{SLC} (SLC-mode wear per P/E cycle, normalized to MLC) into ω_{SLC} (SLC-mode wear per written bit, normalized to MLC). Trivially, knowing that MLC writes two bits per cycle we use

$$\omega_{\text{SLC}} = 2 \cdot \omega'_{\text{SLC}}. \tag{4.13}$$

Correspondingly, we report the ω_{SLC} coefficients in Figure 4.10 and compare their corresponding lifetime with hard partitioning. Remember that, as opposed to the hard partitions lifetime and as described by Equation (4.7), the *Libra* lifetime normalized to an MLC-only device lifetime does not directly depend on the buffer size. Instead, for the same application, a different buffer size would translate into a new SLC-mode cycles ratio, which directly impacts our lifetime extension. Both C1 and C2 curves behave almost linearly and C2 pro-

vides more than 10% extra lifetime for 50% of SLC-mode cycles compared to MLC-only. C1 shows a smaller efficiency and is slower to provide extra lifetime. Nonetheless, both have their ω_{SLC} coefficients smaller than one and thus show a lifetime that is systematically larger than the MLC-only reference. Importantly, evaluating the lifetime from a *fast* experiment can be considered a worst case scenario, where blocks are written at unrealistic frequencies, and provides a conservative lifetime. Still, when we consider significant write ratios to the buffer and for reasonably sized buffers, *Libra* provides up to one order of magnitude more lifetime than hard partitions. In the next section, we stress a simulated flash device from a set of realistic traces in order to evaluate the lifetime extension provided by *Libra* precisely.

4.5 Results

In this section, the proposed soft partitioning technique *Libra* is combined with three published FTLs and the results of running 19 different data traces from three different suites are compared to the hard partitioning architecture.

4.5.1 Experimental Setup

We developed a trace-driven flash simulator in order to measure the execution time and erase counts of several FTL executing realistic traces, whose characteristics are listed in Table 4.2. We generated the *homesrv* and *copy* traces from a tiny server running on a Linux distribution having its root on a flash storage of 16 GB and hosting various standard services (e.g., file server, mail, web). The *homesrv* trace contains one week of this server's system storage activity. The second trace, *copy*, was obtained from writing several GBytes of MP3 files. The next two traces, *fin1* and *fin2*, were obtained from the UMass Trace Repository [3] and produced from OLTP applications running at a large financial institution. The last 15 traces were taken from the MSR Cambridge traces set [48], which contains one week of their data centre activity.

Some of the characteristics of the selected benchmarks are included in Table 4.2. The ratio between memory footprint and total data written gives an indication of the write updates spatial locality. A value close to zero corresponds to a high locality, and a value close to one corresponds to low locality. The average and the standard deviation of the request size indicates how different are the sizes of the different requests. Looking at the traces characteristics, we can conclude that *copy* includes large sequential memory requests and no update locality, *fin2* and prn_0 have a mild update locality, and the rest of the traces have a high locality with memory requests of various sizes. Except for our own traces, most of the traces were gathered from magnetic disks and their file systems were generally not optimized for flash. For example, traces based on disks using a sector size of 4 kB would have a misaligned address space (i.e., sector addresses are not divisible by 4 kB). Accordingly, we realigned the disks address space in order to avoid having 4 kB write accesses systematically overlapping over two flash pages.

Table 4.2: Benchmark Characteristics

Benchmark	Data written [MB]	Footprint [MB]	Footprint ratio	Average request size [kB]	Request size variance [kB]
homesrv	5,566	1,115	0.20	18.6	-13.7/+105.2
copy	3,606	3,598	1.00	395.6	-339.7/+115.3
fin1	14,918	527	0.04	3.7	-1.6/+6.1
fin2	1,860	369	0.20	2.9	-2.0/+10.1
hm_0	20,968	1,670	0.08	8.3	-5.1/+28.4
mds_0	7,542	339	0.04	7.2	-3.8/+10.1
prn_0	47,068	12,683	0.27	9.7	-7.4/+35.2
proj_0	147,729	1,693	0.01	40.9	-31.7/+22.4
prxy_0	55,088	723	0.01	4.6	-3.3/+24.8
prxy_1	742,211	13,078	0.02	13.1	-6.8/+41.3
rsrch_0	11,077	296	0.03	8.7	-4.2/+19.5
src1_2	45,206	669	0.01	32.5	-24.7/+29.4
src2_0	9,563	504	0.05	7.1	-3.6/+9.3
stg_0	15,452	401	0.03	9.2	-5.2/+24.5
stg_1	6,129	405	0.07	7.9	-3.9/+14.9
ts_0	11,611	549	0.05	8.0	-3.6/+21.1
usr_0	13,390	661	0.05	10.3	-5.8/+18.8
wdev_0	7,317	351	0.05	8.2	-4.2/+15.2
web_0	11,952	711	0.06	8.6	-4.2/+20.6

We chose to implement *Libra* for three different hybrid-FTLs, namely FAST [37], ROSE [15] and ComboFTL [26]. FAST is a reference hybrid-FTL, which maps its data partition to the block level. It is light, simple to implement, and suits low-cost storage solutions. ROSE is one of the latest improvements over FAST that we know of. It decreases the garbage collection overhead by using a more advanced and efficient metric to select victim blocks from the buffer. Although both those FTL originally use a regular MLC buffer, we allocate the buffer to SLC, which, as motivated in Section 4.3.1, increases performance and reduces power consumption. The only side effect is that twice as many blocks must be allocated to the buffer, effectively reducing the device capacity, which we assume to pay off for reasonable buffer sizes. Lastly, ComboFTL includes an SLC buffer that gives multiple chances to victim data upon eviction. If the victim data is considered as being likely to be updated, it can be fed back into the buffer avoiding an expensive migration to the MLC partition. ComboFTL has a parameter controlling the write bandwidth directed to the buffer and data partitions. Considering hard partitions, while directing more often writes to the buffer might increase performance it is also likely to reduce the device lifetime, and reversely. Thus, as opposed to the other two FTLs, ComboFTL can tradeoff performance for lifetime. Finally, the ComboFTL address mapping is built on a hierarchy of mappings that provides a thiner granularity than the block level: the data partition is divided in sets of several blocks and each set has its own page-level mapping table. This mapping reduces the garbage collection overhead compared to FAST and ROSE, but requires more RAM to store the translation table.

The flash characteristics considered for the simulation are taken from the C2 chip and we simulated a device of 16 GB. Except for our own traces, most of the considered traces come from disks much larger than 16 GB (up to 1 TB). Still, every trace's footprint is smaller than 16 GB. Hence, when we partition the original disks storage in blocks of the same size as the considered flash (i.e., 2 MB blocks in our case) and consider only the referenced blocks, every trace fits in the simulated device. We perform this compression on the block-level rather than the sector level, in order not to alter data spatial locality, which would have artificially changed the traces simulated behavior. Thereby, the only difference coming from our relatively small simulated device is that the absolute buffer capacity for a 16 GB disk would be smaller than the original disks' buffer. A smaller buffer will incur a higher garbage collection overhead: for the same trace, the victim blocks selected by the garbage collector will more likely contain valid pages to merge into the data partition. Consequently, the data partition will be written more often and the write ratio to the buffer will decrease, which penalizes the soft partition relative lifetime. Note that in Figure 4.10 the lifetime extension of *Libra* monotonically increases with the ratio of writes directed to the buffer. Furthermore, as seen in the hard-partition lifetime model, a write ratio to the buffer that is too high reduces its lifetime dramatically compared to MLC-only. Accordingly, the chosen simulated disk capacity will be advantageous to the hard partitions, underestimating the lifetime extensions that can be achieved with our proposed approach.

4.5.2 Soft vs. Hard Partitioned Hybrid FTLs

The traces are executed by each FTL for several buffer sizes ranging from 1% to 20% of the device's cells. The traces are executed twice: the first run serves as a warm up and we collect the result with the second run. We assume that the data partition is originally fully allocated with valid data, which compared to a typical use case would provide a conservative lifetime for *Libra*. Indeed, a fully allocated device increases the garbage collection overhead and provides a smaller write ratio to the buffer. We visit a large spectrum of parameters specific to each FTL and keep only the most effective combination for each trace. For FAST and ROSE, reducing execution time will systematically maximize lifetime, whereas ComboFTL, originally built on hard partitions, provides parameters to limit excessive writes to the SLC partition to try balancing the wear between the two partitions, trading off performance in the process. Accordingly, we present in our results two parameter sets for ComboFTL: Combo_L maximizes lifetime while Combo_P maximizes performance.

We implemented a classical wear leveling strategy, where we limit the P/E counts difference between the blocks to 100 cycles [61]. Whenever the P/E count difference between a block freshly erased and the block currently having the lowest P/E count exceeds this limit, the former block replaces the latter, which is in turn erased and ready to be allocated. On hard partitions, such wear leveling approach must be separately performed on each partition; instead, for *Libra*, it is performed globally, similarly to a regular MLC-only device. This difference has a very small impact on the lifetime and performance difference between *Libra* and hard partitioning. In order to measure this difference, we executed each traces repeatedly for the wear leveling to be triggered sufficiently. We measured a difference that is systematically below 1% of execution time and lifetime between hard partitions and *Libra*. Thus, the execution time of both the *Libra* and the hard partitioning scheme is assumed the same in our experiments.

Figure 4.11 shows normalized lifetime (top) and normalized execution time (bottom) of the selected FTLs executing the traces of Table 4.2 for a buffer size of 5% of the device's cells. For every combination of FTL and trace, we report the lifetime corresponding to *Libra* (S) and hard partitions (H). The lifetime results are normalized to Combo_P on hard partitions and the execution time is normalized to Combo_P (which is assumed similar for both hard and soft partitions). In this figure, we report the results of *Libra* for C2 only.

We observe that the proposed soft partitioning is able to increase the device lifetime considerably with respect to hard partitioning for the vast majority of the traces and FTLs. When a substantial amount of stress is put on the buffer, Combo_L is able to extend hard partitions lifetime compared to Combo_P, sacrificing significantly the performance. In average, on hard partitions, Combo_L almost doubles the lifetime compared to Combo_P, while increasing the execution time by 25%. Whereas Combo_P on *Libra* quadruples the lifetime on average compared to hard partitions and for similar performance. Interestingly, maximizing lifetime for hard partitions does not improve lifetime for *Libra*. Indeed, Combo_L limits the write bandwidth to the buffer, which reduces the wear of the buffer but significantly increases garbage

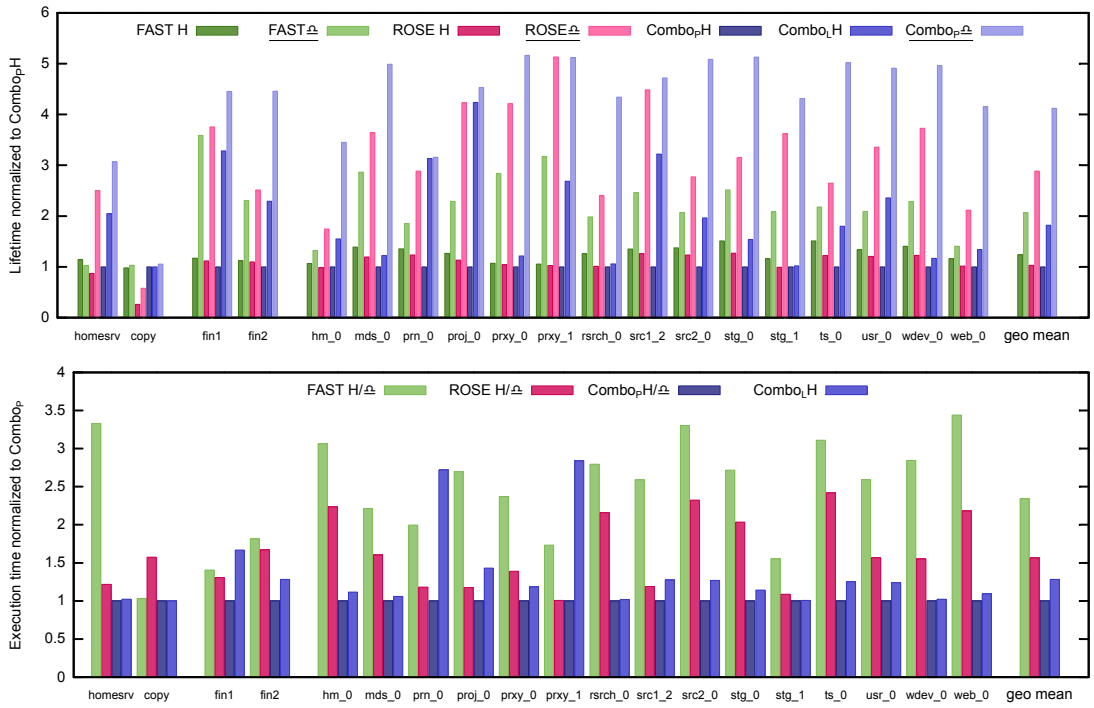


Figure 4.11: **Lifetime and Performance.** The results contrast our technique ('S' versions) versus hard partitioning ('H' versions) for three FTLs implemented with a 5% buffer size and normalized to Combo$_P$ on hard partitions. Among a large spectrum of parameters specific to each FTL, only the best results are shown. Combo$_P$ and Combo$_L$ maximize performance and lifetime, respectively. In the case of performance, we assume negligible difference between hard and soft partitioning. The soft partition results correspond to the ω_{SLC} measured from C2. Overall, our soft partitioning significantly increases lifetime for practically every considered FTL and benchmark.

collection overhead on the data partition. Furthermore, *Libra* benefits from the fact that SLC-erase cycles wear less the cells while improving performance. When hard partitioning requires trading-off lifetime for performance, *Libra* is able to obtain the best of both, therefore the results for Combo$_{\text{L}}$ on *Libra* are omitted from the lifetime results.

The *copy* trace being mostly made of very large sequential accesses, it bypasses completely the buffer and directs most of the accesses to the data partition. Having the majority of writes directed to the MLC partition annihilates most of the benefit of an SLC-MLC combined architecture and it is not surprising to observe similar lifetime between hard and soft partitioning.

4.5.3 Generalization of Experimental Results

Figure 4.12 plots over Figure 4.2 the lifetime results for the different configurations discussed in the previous subsection. New configurations, corresponding to additional log buffer sizes are also added to the figure.

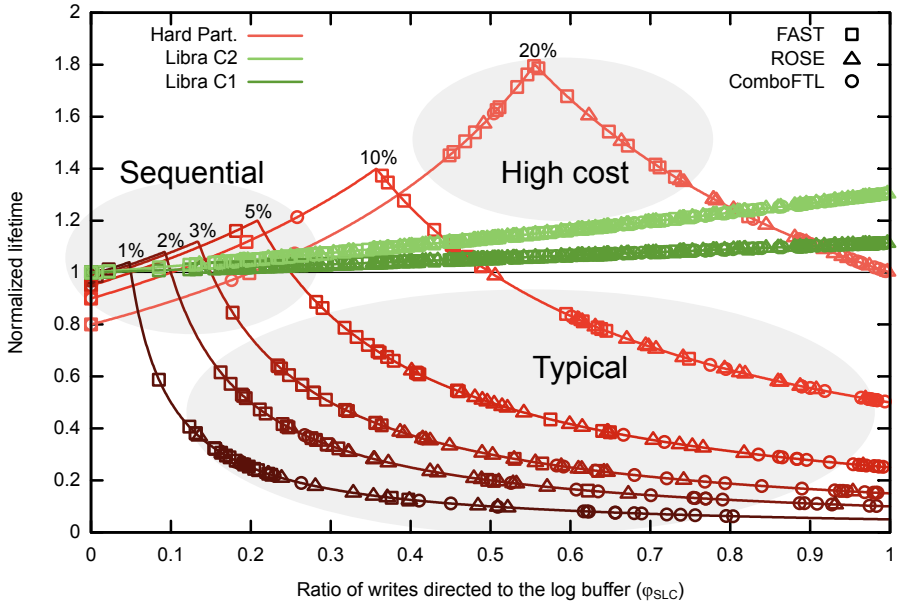


Figure 4.12: **Lifetime models populated by benchmarks results.** This graph shows the result of every best combinations of FTLs, traces and buffer sizes normalized to MLC-only. In typical applications including local updates patterns, soft partitions do systematically better. Only configurations characterized by a considerably higher cost can reach higher lifetimes.

For applications with high update locality, we observe that increasing the buffer size reduces the pressure on the data partition and results in higher ratios of writes to the buffer. This is represented by the data points shifting to the right (i.e., larger write buffer ratio) when going to larger buffer sizes. In that spectrum of the plot, hard partitioning is only able to outperform soft partitioning for very large buffer sizes. Such region is shaded in the figure and annotated as *high cost*. It should be noted that the majority of the points outperforming soft partitioning are from FAST and ROSE, which are, in absolute, less efficient (i.e., worse performance and lifetime) than ComboFTL. When hard partitioning fails maximizing the device lifetime for sensible buffer sizes, *Libra* can extend it by up to 10×. Such region is shaded in the figure and annotated as *typical*. The latter corresponds to scenarios that are likely to target a hybrid device to increase the flash performance and that importantly will largely extend device lifetime when adopting the proposed soft partitioning. For sequential access patterns, all the buffer sizes present a very low ratio of buffer writes, which results in marginal differences between the different cases, except for large buffer sizes. Such region, shaded in the figure and annotated as *sequential*, does not benefit from the hybrid devices schemes targeted in this chapter.

4.6 Related Work

The idea of an SLC-MLC combined architecture has already been investigated in previous work. Grupp et al. [22] experimentally characterize a set of flash chips reporting performance and energy figures, and notice the variable performance in MLC flash between LSB and MSB page programming. Based on those observations, they propose Mango, an FTL that opportunistically skips MSB pages to enhance the responsiveness and reduce the total energy consumed. As opposed to the SLC-mode presented in our work, Mango can bypass an MSB page of a block independently to the other MSB pages. In other words, in Mango, blocks can have valid and bypassed MSB pages at the same time. Consequently, the SLC-mode relative wear that we characterized in our study cannot be considered by Mango to balance the blocks wear, which results in a lifetime reduction.

Chang [11] proposes a hybrid SSD combining SLC and MLC chips, which is a clear example of hard partitioning discussed in this chapter. In order to extend the lifetime, they adapt the ratio of writes directed to the log buffer to balance the wear on each partition. Thereby, in most cases, performance is reduced. Instead, our proposed scheme respects the ratio of writes to the log buffer, which should have been optimized for performance by the FTL, and changes the physical allocation of the log buffer to balance the device wear, obtaining high performance without compromising device lifetime.

Murugan and Du understand the shortcomings of hard partitions in hybrid devices and developed Hybrot [47] accordingly. It uses an integral controller that limits the flow to the SLC partition depending on the workload behavior and the hard partitions dimensions. Still, it uses hard partitions, and with this limited SLC-write flow, the potential benefits from SLC cannot be fully exploited as opposed to the use of soft partitions.

Park et al. [52] propose HFTL, an FTL based on an SSD architecture very similar to Chang's. In particular, they propose techniques exploiting multi-banks parallelism and maximizing bandwidth. As we do too, they realize that the device lifetime is limited by the partition with the shortest lifetime; however, they mitigate the problem by sizing the SLC partition to guarantee a lifetime larger than the MLC partition. This oversizing, with 10 to 30% of the cells allocated to the log buffer, significantly increases the cost of the system, not only for the increase in flash cells but also for the large address translation table associated, which might be prohibitive for some storage classes.

Similarly, Im et al. propose ComboFTL [26], which can be tuned to either optimize lifetime at the expense of reducing performance or performance at the expense of reducing lifetime. Figure 4.11 shows that the combination of ComboFTL optimized for performance with our soft partitions can simultaneously achieve the longest lifetime and the best performance.

Instead of relying on an FTL to interface between a common file system and the flash memory, it is also possible to use specialized file systems that are able to interface the NAND flash interface directly and capable to supervise the wear-leveling and garbage collection. The

JFFS2 [58] is an example of such flash file system and was extended by Lee et al. [38] to build FlexFS, a flash file system that enhances the storage responsiveness by selectively choosing to write data into SLC-mode or MLC-mode depending on the device's capacity available. Unlike the hybrid devices above-mentioned, both cold and hot data are stored in SLC-mode, which increases significantly the garbage collection overhead and consequently sacrifices the device's lifetime. Furthermore, the wear of SLC-mode cycles are assumed equal to regular MLC-mode cycles, which prevents to exploit the flash endurance to its fullest. With a little effort, FlexFS could be adapted to implement the mechanisms of *Libra* to balance the mixed wear and make a better use the device's endurance.

To the best of our knowledge, *Libra* is the first work that introduces a soft SLC/MLC partitioning of the log buffer present in hybrid FTLs; with it, the log buffer is continuously reallocated to distribute the device wear and thus extending the device lifetime at virtually no cost.

4.7 Conclusions

Flash architectures combining SLC and MLC technologies are targeting new cost-sensitive applications with large data update locality. Frequent updates benefit from the superior SLC performance while devices are primarily in MLC-mode to take advantage of the lower cost of MLC devices. However, unbalanced pressure on the SLC partition may lead to a premature death of the device. In this chapter, we have presented *Libra*, an approach that is robust to unbalanced stress. Using data extracted from measurements on actual flash chips, and making conservative estimations, *Libra* shows a lifetime at least as long as that of an MLC-only device and shows up to 10 times longer lifetime compared to known SLC-MLC approaches. Furthermore, this advantage comes at practically no extra cost and without any performance loss, which is particularly interesting for high-volume consumer products.

In this chapter, we have characterized the SLC-mode, a feature neglected from typical specification documents. With this characterization, we did not only validate the SLC-mode usage in our system, but we also found that it could extend the flash storage lifetime compared to MLC-only. These findings contradict previous work relying on SLC-mode [22, 38] that assumes a similar wear between SLC and MLC, and confirm the importance of characterizing physical properties neglected by manufacturers before making use of them. In the next chapter, we will focus on the block endurance variance, another characteristic of flash memory that is neglected from specification documents.

5 Phœnix: Reviving MLC Blocks as SLC

In this chapter, we will concentrate on the variance of a NAND flash block endurance. This is typically a property neglected from flash manufacturer, as their aim is to provide a device that looks as uniform as possible. Yet, in this chapter, we will characterize the variance in block endurance for a flash chip, propose a model to describe the block endurance distribution and present a technique that reuse the first blocks wearing out to relax the pressure on the remaining healthy blocks. Specifically, in this chapter we present Phœnix, a strategy reviving bad MLC blocks as SLC to extend NAND flash lifetime.

5.1 Introduction

As already discussed in the previous chapters, continuous pressure is put on NAND flash technology to push it towards higher densities, which comes with lower performances and with a severe hit in endurance. Interestingly, flash memory degrades progressively and despite the use of efficient wear-leveling techniques, some blocks will fatally wear out earlier than others will. In this chapter, we present Phœnix, a novel scheme to extend current FTL that mitigates the degradation in lifetime of MLC flash. Essentially, we build on the SLC/MLC soft partitioning architecture presented in Chapter 4 and propose to keep on using worn-out MLC blocks as SLC blocks. By 'reviving' these blocks, which could appear pointless at first sight, we show that the lifetime of current flash devices can be extended by up to 17% at no cost. Furthermore, as flash goes to smaller technology nodes and cell bit-density increases, we should expect a relatively larger endurance variance, which let us envision lifetime extension.

5.2 Reviving Bad Blocks

As discussed in Section 2.3, programming and reading a single bit in a cell is more reliable than multiple bits. Thus, SLC can generally experience one and two order of magnitude more P/E cycles than MLC and X3 MLC, respectively [23]. Interestingly, we have seen in Chapter 4

that typical MLC can also be managed as SLC by storing a single bit [54, 23, 28, 30]. Thereby, in this SLC-mode, the MLC experiences similar performance, energy, and endurance benefits of SLC technology. An MLC block will be considered unreliable by the FTL when its BER approaches dangerously the limits of the ECC unit. Yet, this block could still be used in SLC-mode, which would show a significantly lower BER and be usable for a significant number of extra P/E cycles. Indeed, as discussed in Chapter 2, programming a single level enables larger voltage threshold margins and generates significantly less cell-to-cell interference. In this chapter, we propose a method using this feature to extend the lifetime of MLC flash storage devices.

5.2.1 Reviving MLC Blocks in SLC-mode

Typical flash devices stop using a block whenever it permanently fails (e.g., circuit defects) or when it becomes unreliable by showing a bit error rate that is too high to be handled by the implemented ECC. For this reason, flash manufacturers suggest to reserve a set of extra blocks (e.g., 2% of the total capacity [44]) as spare blocks to replace the "bad" ones. Such a device will be considered dead when running out of spare blocks.

While permanent failures might prevent any further use of a block, we propose to revive the unreliable blocks with high error rates by using them in SLC-mode. The SLC-MLC hybrid devices built on soft partitions and presented in Chapter 4 will serve as a baseline device. Revived blocks will be allocated to the log buffer partition, where blocks typically take a greater amount of incoming writes than data blocks. Thereby, revived blocks are not only sparing the use of free blocks, they also reduce the stress applied on the remaining healthy blocks.

Revived blocks can be managed in a very similar way to typical bad block management. A list of bad block is generally maintained in some reserved area of the flash device and can easily be extended with a flag to differentiate bad blocks from revived ones, for a negligible cost. A similar flag could redundantly reside in the FTL memory in order to differentiate a healthy block from a revived one rapidly. Thereby, reading those redundant flags avoids accessing the flash and prevents any identification-time overhead. Storing this redundant flag in memory would require a single bit for every free block and block allocated to the buffer, and thereby, comes with practically no extra cost.

Figure 5.1 illustrates the difference between a baseline device implementing a typical hybrid-FTL block management policy and the proposed Phœnix technique during their lifetime. Both initially allocates four different partitions: a data partition of size ρ_D, a buffer partition of size ρ_B managed in SLC-mode, a set of ρ_F free blocks, and a set of bad blocks that we assume empty at the beginning of the device life. Throughout its lifetime, the baseline device will regularly identify blocks as broken or unreliable and move them to the bad set, gradually emptying the free set. When the device runs out of free blocks, it is considered dead. This is illustrated in Figure 5.1 by the last state of the baseline example (bottom left).

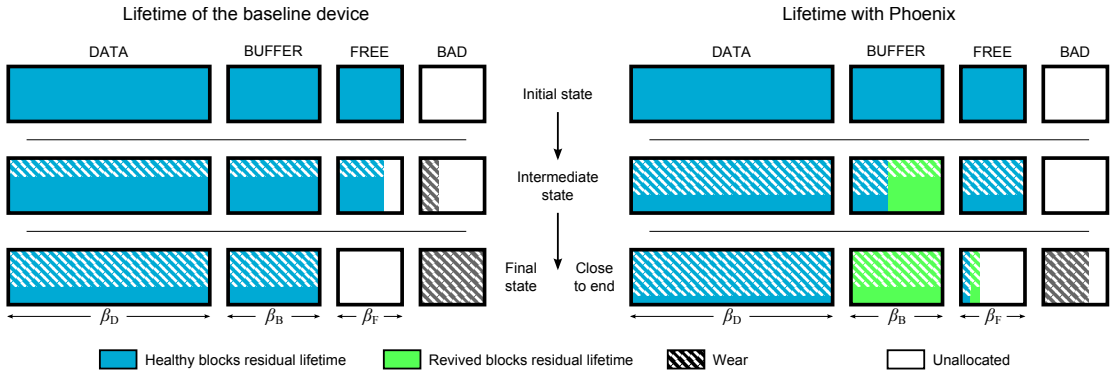


Figure 5.1: **Phœnix compared to the baseline device.** Each device starts with a data, a buffer and a free set of size ρ_D, ρ_B and ρ_F, respectively and a bad set initially empty. While the baseline approach discards unreliable blocks to the bad set, Phœnix revives them as SLC and uses them in the buffer. Reviving a bad block extends its lifetime and reduces the stress on the remaining healthy blocks. In the end, Phœnix will benefit SLC-mode lifetime, while better exploiting the MLC lifetime.

When using Phœnix, the buffer and the free set can now allocate both revived and healthy blocks, while the data partition is still restricted to healthy blocks. Blocks that are detected as unreliable for MLC are labeled as *revived* and are kept in the free set. (Permanently broken blocks are directly moved to the bad set.) Figure 5.1 shows how healthy blocks get progressively replaced by revived blocks in the buffer. Now, less healthy blocks are required for the device to stay alive which results in a longer device lifetime.

With Phœnix, the buffer would ideally allocate only revived blocks maintaining the device alive as long as enough healthy blocks are available for the data partition. Whenever the buffer needs to allocate a new block from the free set, it will give priority to the revived blocks in order to minimize the stress on the healthy blocks. Thus, a block will only be dropped into the bad set when it is considered unreliable in SLC-mode. Although Phœnix can perfectly be used by any FTL (including page-level FTL) mixing SLC-mode and MLC, here we focus on its implementation on hybrid FTLs for the same reasons listed in the previous chapter. In the next section, we discuss in detail the models evaluating quantitatively the lifetime improvements that can be expected from Phœnix. These models assume a hybrid FTL, where the number of blocks allocated in SLC-mode are constrained by the buffer size, where as for page-level mappings this number can fluctuate at run time.

5.3 Device Degradation Models

We measure empirically the block endurance distribution from a real NAND flash chip. Based on this data, we propose two models to describe and confront the lifetime of a baseline device against Phœnix.

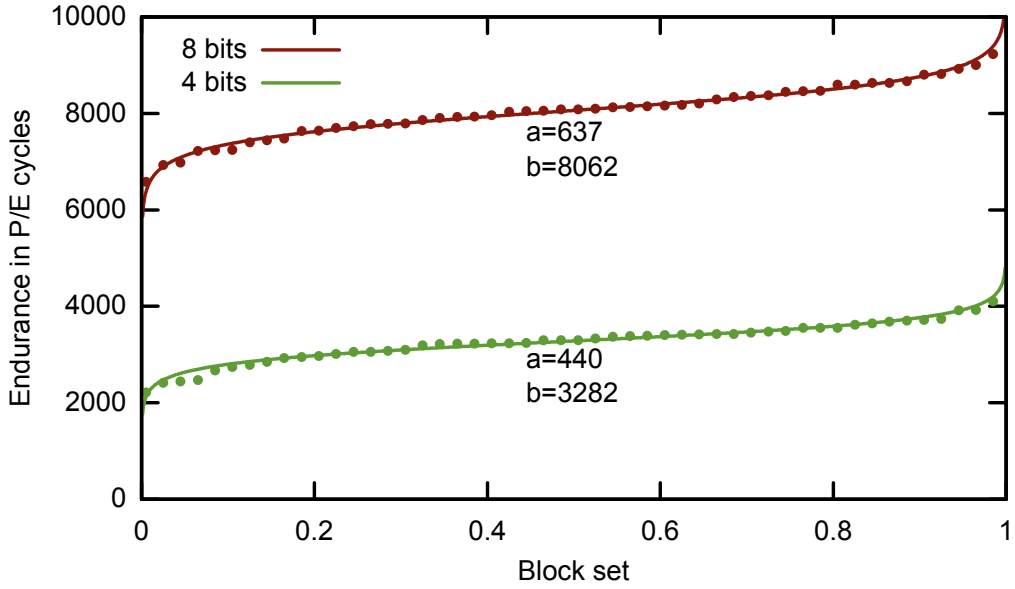


Figure 5.2: **Block endurance measured in a real NAND flash chip.** We report the endurance of a set of blocks measured from a real NAND flash chip, assuming four and eight faulty bits as error thresholds. The blocks are ordered from the smallest endurance (on the left) to the largest (extreme right). Each solid line is the model function of Equation 5.1 fitted to the measured data (markers). For each, we provide the corresponding parameters a and b.

5.3.1 Block Endurance Distribution

Our approach relies on the fact that the blocks do not all die after exactly the same number of P/E cycles. Otherwise, every successive write to the data partition happening after the first worn-out block would result in a new worn-out block; therefore, the lifetime extension obtained from reviving those blocks would be virtually zero for typical use-cases.

Accordingly, we run a synthetic test on a set of 50 blocks of an MLC NAND flash chip. The characterization experiment correspond to the typical Erase, Program, and Read cycles described in Chapter 3: for each cycle, each block is erased, then programmed with random data, and finally read back and checked for correctness. The test monitors the evolution of erroneous bits per page for several thousands of cycles.

A particular block is considered to be unreliable whenever a given error threshold is reached in any of its pages. The actual threshold will depend on the ECC capabilities of a particular device. A stronger ECC extends significantly the lifetime of a block but requires hardware that is more complex, increases access latency, and adds some storage overhead. Accordingly, Figure 5.2 plots the block endurance cumulative distribution measured for 4- and 8-bit error per page thresholds. The chip is organized in blocks made of 128 pages of 8 kB and corresponds to C1 of Table 4.1.

We fit the measured cumulative endurance distribution with the following inverse hyperbolic tangent function:

$$f(\theta) = a \cdot \operatorname{artanh}(2 \cdot \theta - 1) + b \quad \text{for} \quad 0 < \theta < 1, \tag{5.1}$$

with θ representing a proportion of blocks, $f(\theta)$ the largest endurance in P/E cycles within the θ weakest blocks, and a and b being the parameters of the distribution. Parameter b corresponds to the average endurance, while a is function of the variance. We provide the parameter set for each fitted curve in Figure 5.2.

5.3.2 Analytical Model of Baseline Device Lifetime

Next, we present an analytical model to compute the expected lifetime of a typical flash device given its block endurance distribution and it partitions size. We refer to lifetime as the total amount of data that can be written in a block or a device before wearing it out. In Chapter 4, we studied the MLC- and SLC-mode mixed usage and observed empirically that programming a block once in MLC or twice in SLC-mode applies practically the same wear to the block resulting in very similar device lifetime. Conservatively, and in order to simplify the result readings, we assume an equal wear per written bit for SLC- and MLC-mode.

Let θ_t be the maximum ratio of blocks that can wear out for a flash device before it dies. It typically represents the maximum size of the bad block set. Assuming a perfect wear-leveling, the integral of $f(\theta)$ from 0 to θ_t gives us the lifetime exploited by weakest blocks, while the remaining healthy blocks are limited to $f(\theta_t)$ cycles. Hence, the MLC lifetime component L_{MLC} of a flash device is

$$L_{\text{MLC}}(\theta_t) = [F(\theta)]_0^{\theta_t} + f(\theta_t) \cdot (1 - \theta_t), \tag{5.2}$$

with $[F(\theta)]_0^{\theta_t}$ being the integral on the weakest blocks and $f(\theta_t) \cdot (1 - \theta_t)$ being the lifetime exploited by the remaining blocks. This lifetime is illustrated in Figure 5.3 by the surface of the dark area. The model assumes a perfect wear-leveling algorithm that evens the P/E counts of every healthy blocks, which keeps the variance in endurance to lower values than what we could expect in reality. A larger variance would benefit the presented strategy as it will lower $f(\theta_t)$ and let the weakest block to be revived sooner. Hence, this assumption produces conservative results.

The baseline device reaches its lifetime limit when ρ_{F} blocks wear out. As its lifetime is limited to the MLC lifetime, it is equal to L_{MLC} with $\theta_t = \rho_{\text{F}}/\rho$, and with ρ being the total number of block in the device. Therefore its lifetime L_{B} becomes

$$L_{\text{B}} = L_{\text{MLC}}(\frac{\rho_{\text{F}}}{\rho}). \tag{5.3}$$

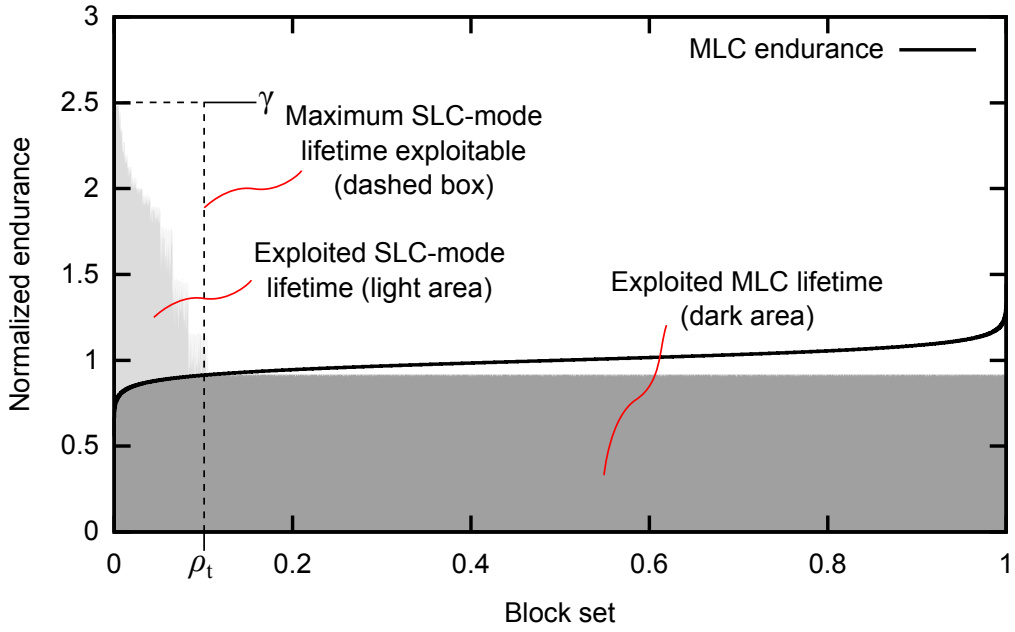


Figure 5.3: **Lifetime breakdown.** The solid line shows the block MLC endurance normalized to the average. The dark region represents the exploited MLC lifetime given a maximum θ_t bad blocks. The light region is an example of the additional SLC-mode lifetime that could be exploited by Phœnix. Using the maximum SLC-mode lifetime delimited by the dashed box is challenging, because, as more blocks get revived, the time to exploit them gets shorter.

5.3.3 Analytical Upper Bound of Phœnix Device Lifetime

In order to evaluate the lifetime extension brought by Phœnix, we must describe a relationship between the lifetime of MLC and SLC-mode. We use a parameter γ such that $\gamma \cdot l_M = l_S$, with l_M and l_S being the lifetime of MLC and SLC-mode, respectively. While l_M is provided in manufacturer datasheets, l_S is typically not known. We setup a simple experiment to attempt to extract this value, and did find a very large difference between the MLC and SLC-mode lifetimes with γ getting typically larger than five. However, this experiment was too simple and did not cover all the parameters such as data retention. The retention characteristics being significantly different between SLC and MLC modes, a thorough evaluation of the SLC versus MLC endurance would require significantly more efforts. Therefore, rather than trying to setup this delicate experiment, we prefer to rely on data, pessimistic in our sense, provided by Im and Shin [26], who relate the endurance for a specific flash device that explicitly provides SLC and MLC modes. Specifically, the device blocks could sustain either 10,000 MLC-erase cycles or 50,000 SLC-erase cycles. Therefore, given that MLC writes twice as many bits per P/E cycles compared to SLC, a block in SLC-mode could be written 2.5× more than in MLC. Thereby, for our experiments, we assume γ to be 2.5.

While the baseline lifetime is bounded by $\theta_t = \rho_F/\beta$, Phœnix extends this limit with $\theta_t = (\rho_F + \rho_B)/\beta$. Furthermore, Phœnix revives the weakest θ_t blocks, which can now expect a

maximum endurance of γ. Hence, the theoretical maximum Phœnix lifetime corresponds to the union of the dark area surface with the dashed box from Figure 5.3 and is equal to

$$L_{\text{Pmax}}(\theta_t) = \gamma \cdot \theta_t + f(\theta_t) \cdot (1 - \theta_t). \tag{5.4}$$

Referring to Figure 5.3, for $\theta_0 = \rho_F + \rho_B$, $\gamma \cdot (\rho_F + \rho_B)/\rho$ would correspond to the surface to the left of θ_0, while $\rho_D/\rho \cdot f^{-1}((\rho_F + \rho_B)/\rho)$ would be the surface on the right.

A factor limiting the theoretical maximum potential of Phoenix is the buffer utilization frequency. Revived blocks are exclusively allocated to the buffer; therefore, when it is underutilized, the extra lifetime provided by the SLC-mode cannot be exploited. Accordingly, we define another bound to the maximal reviving lifetime, $L_{\text{Pbuf}}(\rho, \theta_t)$, that is function of a ratio ρ of writes to the buffer with

$$L_{\text{Pbuf}}(\rho, \theta_t) = \frac{L_{\text{MLC}}(\theta_t)}{1 - \rho}. \tag{5.5}$$

This function returns the MLC lifetime plus the total number of writes to the buffer, which corresponds to the maximum SLC-mode lifetime exploitable. Combined with L_{Pmax}, we derive $L_{\text{Pbound}}(\rho, \theta_t)$, the global upper bound, which is the minimum of both functions,

$$L_{\text{Pbound}}(\rho, \theta_t) = \min(L_{\text{Pmax}}(\theta_t), L_{\text{Pbuf}}(\rho, \theta_t)). \tag{5.6}$$

This upper bound is plotted in Figures 5.4 and 5.5 for two different device configurations that we will discuss later. This bound is still ideal and very difficult to reach in practice: it disregards any sequentiality constraint in the accesses into the buffer and data partitions. Specifically, this ideal bound assumes that the full potential of SLC-mode lifetime can be exploited before the end of the device's lifetime, which in practice is unlikely as we start exploiting this SLC-lifetime when blocks are turning bad and the devices approaches the end of its lifetime. Thus, in the next section we produce the expected lifetime gain from simulations that evaluates a more realistic scenario.

5.4 Results

In this section, we show how Phœnix behaves on a simulated FTL executing the same disk traces than presented in Chapter 4. We extended ROSE and CombotFTL with Phœnix in order to evaluate it with our flash simulator. At the beginning of the simulation, the endurance of every block is randomly set according to the distribution of Equation 5.1, with parameters $a = 637$ and $b = 8062$. We repetitively input the same trace until the simulated flash device eventually dies. For traces with small data footprint, the hot data would eventually fit the log buffer and bias the hot write ratio to higher values. In order to prevent this, we only used a subset of the traces with sufficiently large data footprint, namely *homesrv*, *copy*, hm_0, prn_0, $proj_0$ and $prxy_1$. We generate results for two different buffer/free set sizes (2%/2% and 5%/5%) and for variants with or without Phœnix.

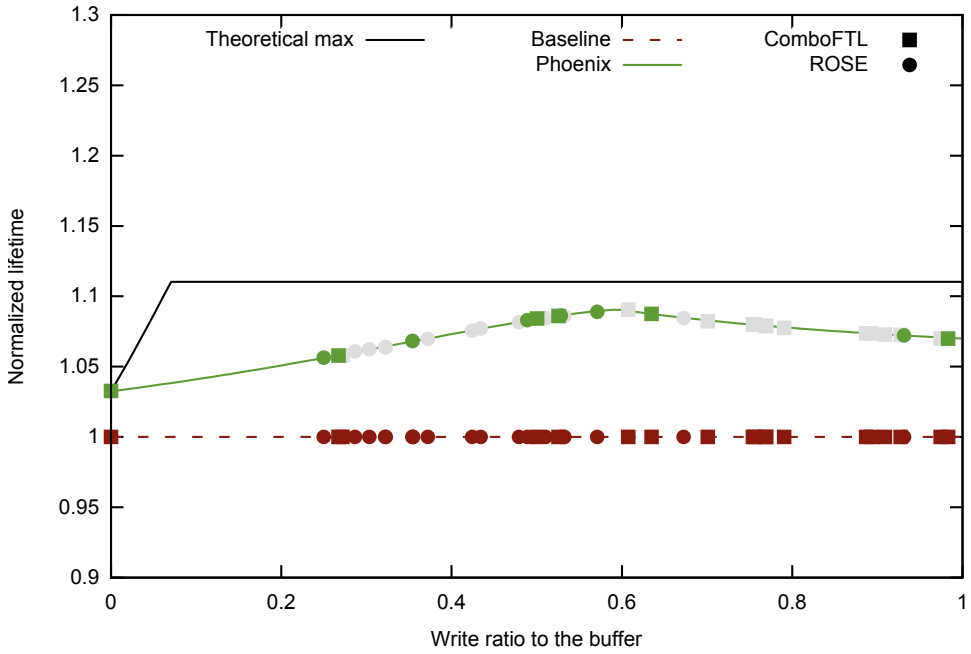


Figure 5.4: **Lifetime comparison with 2% buffer size.** The configuration for this figure uses a buffer and a free set of 2% the total capacity each. The lifetime is normalized to the baseline. The theoretical maximum achievable with Phœnix for the given setup is represented by the black solid line. Although the efficiency of Phœnix depends on the amount of writes directed to the buffer, it is systematically larger than the baseline. The green points are acquired by executing continuously large traces until the simulated device is completely worn out. The gray points are simply projected to the model from a single execution of small traces.

Phœnix does not hamper the performance of the FTL, actually it reduces slightly the wear-leveling overhead when approaching the end of the device lifetime, which increases marginally the performance by less than 1%. From our simulations output, we report the ratio of writes to the buffer, the number of time that each block was programmed and the corresponding device lifetime. In Figures 5.4 and 5.5, the green data points on the top curve represent the full simulation results from the traces selected earlier, while the curves are generated from the simplified simulations that assume a constant ratio of writes to the buffer. We systematically measured an error lower than 0.1% between the full and simple simulation. Therefore, for a specific application or FTL, if the average buffer write ratio is known, we can quickly get a good estimate of the lifetime outcome. In the figures, the gray data points correspond to projections on this model for the remaining traces, for a single trace execution. We see in Figure 5.5 that up to 17% lifetime extension is achieved compared to the baseline.

In Figure 5.3, we illustrate the final state of the simulated device configured as 5%/5% after executing the *homesrv* trace on the ROSE FTL. The surface of the light gray area on the left represents the total SLC-mode lifetime that could be exploited by Phœnix, in this case,

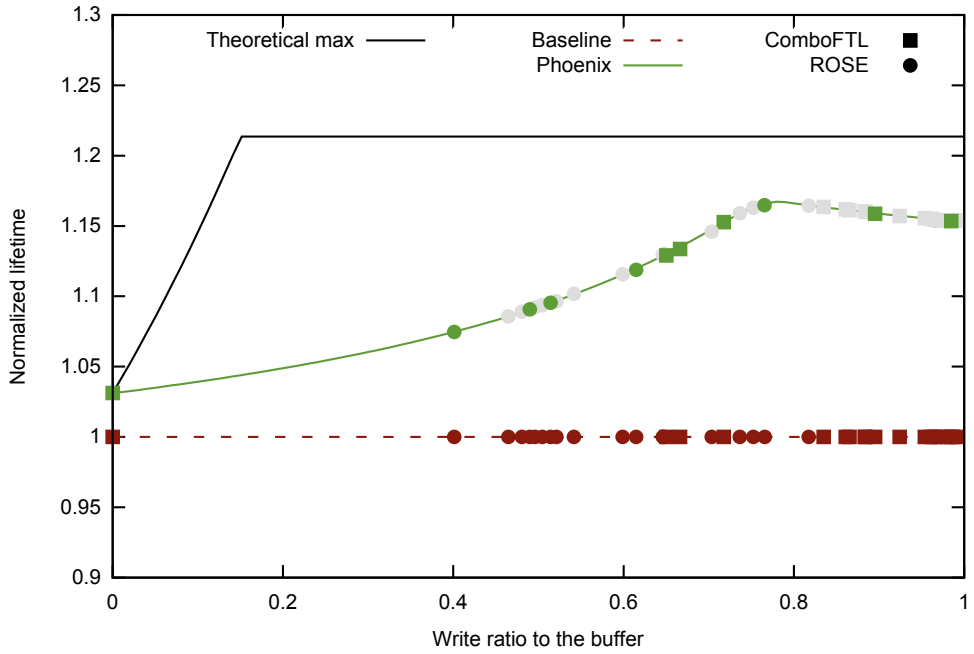


Figure 5.5: **Lifetime comparison with 5% buffer size.** The configuration for this figure uses a buffer and a free set of 5% the total capacity each. A larger free set provides more blocks to revive and leaves more time to exploit their extra lifetime.

roughly half of the theoretical maximum, which translates into a 12% improvement. While the current improvement might be humble, we know that more variability in the block endurance amplifies the potential of our scheme, which will inevitably be happening with the future technology nodes.

5.5 Future Perspectives

The manufacturing process of future MLC and TLC chips will suffer from less accuracy [23], which translates into larger quality variance among the cells. A larger variance results in a smoother degradation of the device, which leaves more time to exploit SLC-mode extra lifetime and benefits our scheme. We propose in Figure 5.6, a set of lifetime extension models obtained with Phœnix for a growing variance qualified by the ratio between the parameters a and b of the distribution function of Equation 5.1. We also show the expected lifetime for a value of $\gamma = 8$, which we assume corresponds to the lifetime relationship between TLC and SLC-mode. One clearly see that increased process variability will make the idea of this chapter more relevant: while the reference lifetime will decrease with new denser technology node, our technique could easily bring 50% more lifetime for free.

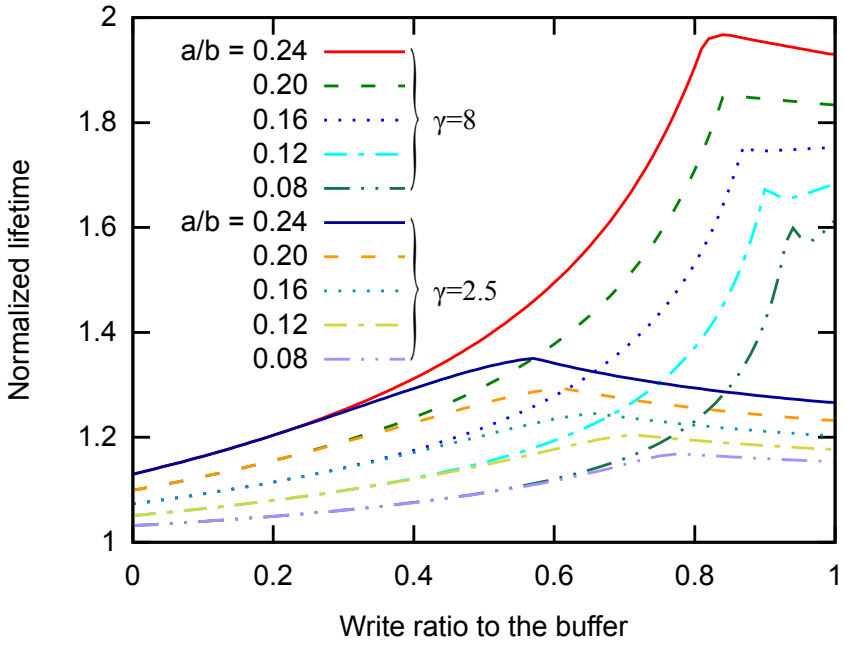


Figure 5.6: **Exploring lifetime gain for future technology.** We plot the expected lifetime when using Phœnix normalized to its baseline counterpart, built on a 5% buffer size and 5% free size. We vary the ratio between the distribution parameters a and b and the γ factor to represent the increasing process variability as new technology nodes will be used for flash manufacturing. The parameters $a/b = 0.08$ and $\gamma = 2.5$ correspond to the results presented earlier. The figure shows that, in the years to come, the tolerance to variability may become critical to achieve useful lifetimes.

5.6 Related Work

So far, most of the efforts invested to extend the lifetime focus on enhancing wear-leveling techniques or reducing garbage collection overhead by improving the way FTLs allocate data. Each of those techniques implemented on FTLS relying on an SLC-mode buffer and MLC data can benefit from our technique.

To the best of our knowledge, the only proposal that extends flash lifetime reusing bad blocks is provided by Wang and Wong [56]. The authors observed that when a block is considered bad, most of its pages are still healthy. They propose to combine the healthy pages of a set of bad blocks together to form a smaller set of virtually healthy blocks. This technique requires storing extra data structures to keep track of a limited set of bad pages; therefore, it comes for a cost. The authors did not provide any information about the block endurance variance that they used to evaluate their strategy. Therefore, they were not able either to quantify clearly the lifetime extension coming with their technique.

Other pieces of work [50, 57] acknowledge the block endurance variance and proposed strategies to address it by putting more load on the strongest blocks. Those techniques require means to estimate the endurance remaining for each block, which is not as trivial as the authors might suggest. These will be better covered and confronted in the next chapter.

5.7 Conclusion

NAND flash cell storage reliability becomes challenging as cells get smaller and more bits are written to them. In this chapter, we presented Phœnix, a technique that revives bad blocks using the fact that cells in unreliable MLC blocks can still reliably be used to store a single bit per cell. This technique does not cause any direct or indirect extra cost: interestingly, it even reduces slightly the wear-leveling overhead when the device reaches its twilight. We presented a simple approach to estimate the lifetime extension that can be expected with our scheme given a benchmark and FTL main behaviors. Phœnix can easily be implemented on top of any existing hybrid FTL and does not require any additional resource—hence, any lifetime benefit comes for free. Using actual flash chip characteristics and a set of conservative assumptions, we showed up to 17% lifetime extension and we are convinced that future chip technology will inevitably bring more variance in the block endurance, which will significantly amplify the advantages of the presented contribution.

In this chapter, rather than characterizing a global device property, we collected and made good use of the statistics on the variance of block endurance in order to build a model for flash devices degradation. Although those statistics might be known by the manufacturer, they are generally omitted from specification documents. Yet, they are essential to qualify strategies such as Phœnix and any related work that would address the endurance variance. In the next chapter, we will exploit the variance in endurance at a different level: we will break the conventional way of accessing flash memory and thus our novel technique will require careful validation through characterization.

6 Wear Unleveling: Relieving Weak Pages to Balance Endurance

In the previous chapter, we proposed a strategy to address the block endurance variance. For this chapter, we characterize the endurance variance on the page level and observe a larger variance than for the block level. Paradoxically, the strategy presented in the previous chapter increases the load on the weakest blocks. This time, we will propose the opposite and reduce the load on the weakest pages. However, producing a precisely unbalance load so that pages within a block degrades more gracefully is not straightforward, and requires a careful characterization to validate any strategy using this approach.

6.1 Introduction

In this chapter, we present a technique to extend flash devices' lifetime that can be adopted by any FTL mapping the data at the page level or at any finer granularity (e.g., sector level). It is also suitable for hybrid mappings [37, 16, 39, 15], 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 6.1). 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 for the flash device to provide its full capacity as long as possible. 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

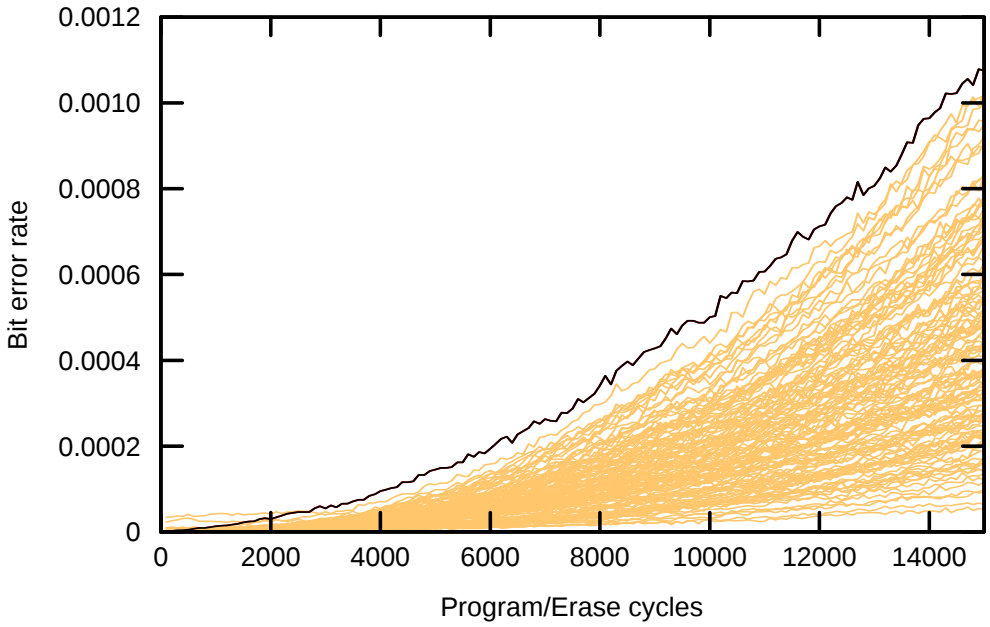


Figure 6.1: **Variance in the degradation speed of different pages.** These data were generated by continuously writing random values into the 128 pages of a single block of flash. The graph illustrates how the BER can grow at widely different speeds among pages of the same block. We suggest reducing the stress on the weakest pages in order to enhance the block endurance.

lifetime extension in future flash memories. This technique can also be used to relax the strength of ECCs, when the required resources and the time to decode such codes might become prohibitive.

6.2 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.

6.2.1 Definition

We define a *relief* cycle on a page the fact of keeping the corresponding cells at the erased state between two erase cycles or, in other words, not programming the page 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 6.2.3. 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

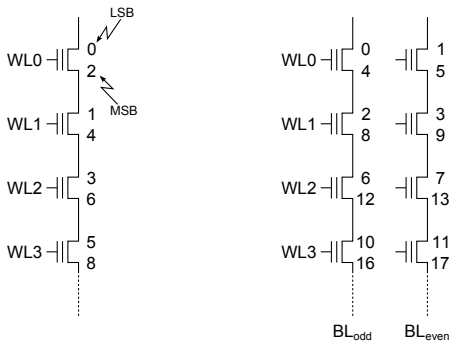
a P/E cycle is correlated to the amount of charge injected for programming; of course, more charges means more damage to the cell. Therefore, a page will experience minimal damage during a *full* relief cycle while a *half* relief cycle will apply a stress level somewhere between the *full* relief and a normal P/E cycle.

We characterize the relief mechanism and quantify its effects in the next sections, where we measure the page degradation speed depending on the relief rate on two real NAND flash chips.

6.2.2 Page Endurance

While accumulating P/E cycles, a block becomes progressively less efficient in the retention of charges and its BER increases exponentially. The endurance, or the point in time when a block will be considered unreliable, depends generally on the following factors: First, the cell design and technology will define its resistance to stress; this is generally a trade-off with performances and density. Second, the endurance is associated with a retention time, that is, how long data is guaranteed to remain readable after being written; for a longer retention time requirement, relatively healthy cells are necessary, which limits the corresponding 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 6.1, the chosen strength will only be justified for a minority of pages while degrading the latency of every page access. Our proposed balancing of page endurance within a block will delay the BER degradation of the weakest pages; therefore, our idea can be used either to reduce the ECC strength requirement or to extend the device lifetime. However, in this chapter, we will only explore the impact of our technique in device lifetime extension.

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 6.1, 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 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.



(a) ABL (b) HBL

Figure 6.2: **Flash cells organization.** Figures (a) and (b) show two examples of cell-to-page mappings in 2-bit MLC flash memories. Each bit of an MLC is mapped to a different page. For instance, in Figure (a), the LSB and MSB of WL_1 are mapped to pages 1 and 4, respectively. The page numbering also gives the programming order; therefore, prior to programing the MSB of a WL, the LSB of the next WL is programmed. This cell programming *cross-sequence* narrows the disturbance that occurs during programming. Figure (b) presents another MLC architecture, where the even and odd pages form two interleaved groups of LSB and MSB pages, making four pages per word line. We evaluated one chip for each of these mappings in our experiments.

6.2.3 Understanding the Relieving Effect

In order to characterize the effects of relieving pages, we reuse the MLC chips introduced in Chapter 4. We remind their respective characteristics in Table 6.1. 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 6.2(a). The C2 chip implements the *Half-Bit Line* (HBL) architecture illustrated in Figure 6.2(b). Specifically, compared to HBL, ABL uses every bit line in parallel. Accordingly, for the same number of bit lines, it roughly doubles bandwidth for a larger control logic and an increase in latency.

We design an experiment to measure on our flash chips how the relief rate influences 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., not any 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

Table 6.1: 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	HBL

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

Figures 6.3 and 6.4 show 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:

$$E = \frac{1}{\omega}. \tag{6.1}$$

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:

$$E_X(\rho,\alpha) = \frac{1}{(1-\rho)\omega + \rho\alpha\omega} = \frac{E}{(1-\rho)+\rho\alpha}. \tag{6.2}$$

Assuming a maximum BER of 10^{-4} to define a page endurance, we show in Figure 6.5 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 (6.2) 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 re-

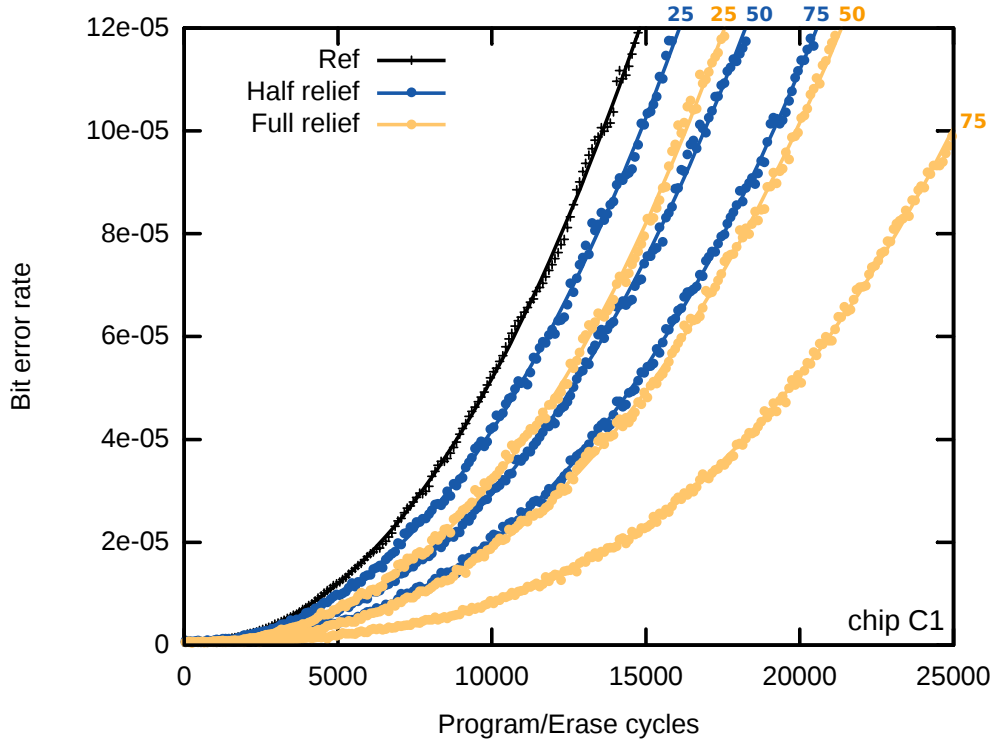


Figure 6.3: **Measured effect of relieving pages on C1.** The degradation speed for various relief rates and types are measured on both chips. The *Ref* curve reports the BER of the entire reference blocks, whereas for the relieved blocks, the BER is only evaluated on the relieved page. The labels '25', '50', and '75' indicate the corresponding relief rate in percent. The BER is evaluated over a 100-cycle period.

lief 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:

$$2 \cdot \alpha_H = 1.22 < 1.39 = 1 + \alpha_F. \tag{6.3}$$

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 6.1). Thus, a half relief is more efficient for the same amount of written data and additionally displays better performance.

Figure 6.6 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 a 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.

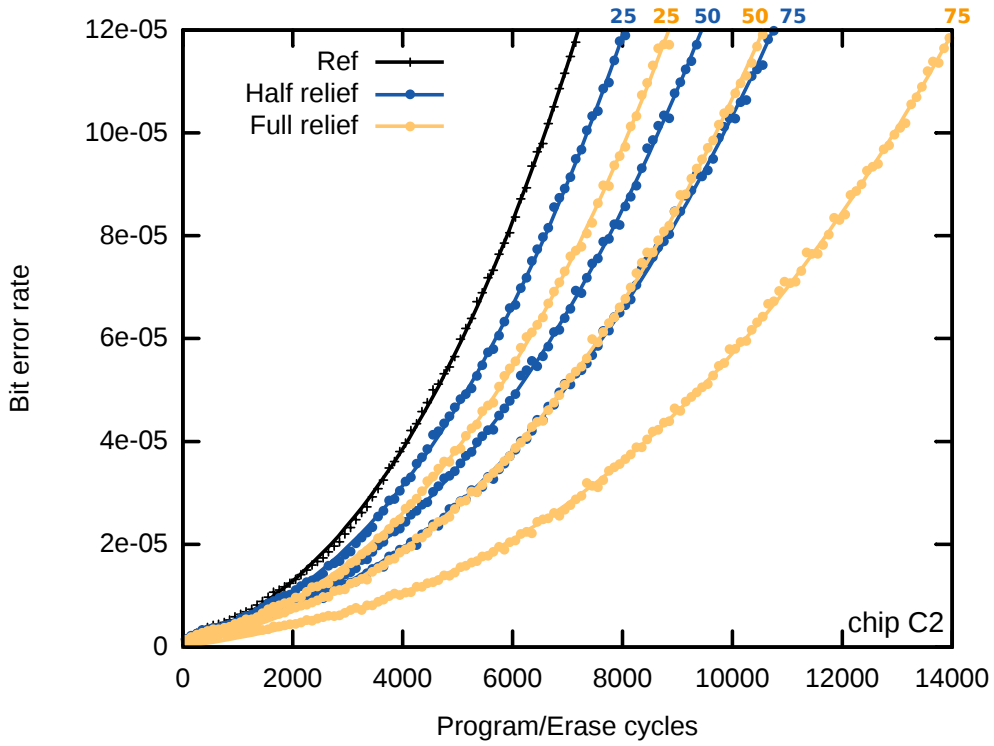


Figure 6.4: **Measured effect of relieving pages on C2.**

6.3 Implementation in FTLs

In this section, we describe the implementation details required to upgrade existing FTLs with our technique to relieve weak pages.

6.3.1 A New Page State

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—as seen in Section 2.4, typical states are *clean*, *valid*, or *invalid*, also illustrated in Figure 6.7. To enable our technique, we introduced a fourth page state, *relieved*, to indicate pages to be relieved (i.e., not be 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. Besides, we even observed that relieving pages significantly reduces the amount of faulty bits in neighbors for the corresponding P/E cycles.

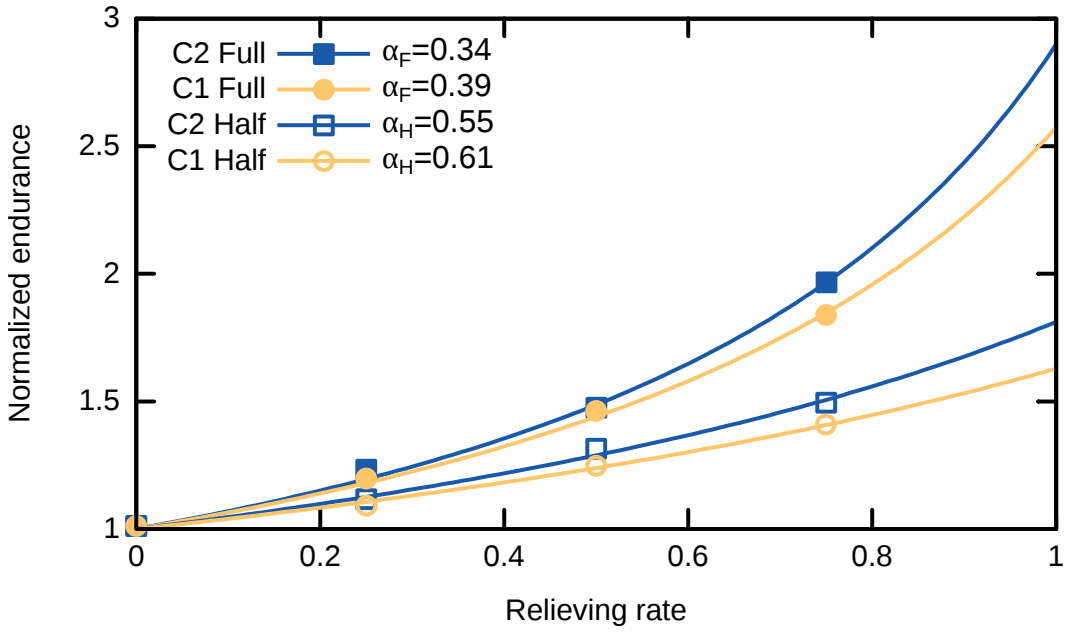


Figure 6.5: **Normalized page endurance vs. relief rate.** The graph shows how relieving pages extends their endurance. The endurance is normalized to the normal page endurance, corresponding to a maximum BER of 10^{-4}. For each chip, the relative stress of the full and half relief type is extracted by fitting the measured points.

6.3.2 Mitigating the Capacity Loss

Relieving a set of pages during a P/E cycle temporarily reduces the effective capacity of a block. Therefore, applying this strategy for 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. Furthermore, in order to limit the total capacity loss while still being able to relieve pages frequently, we propose to enable relief cycles exclusively in blocks that are allocated to the hottest partition, where the FTL writes data identified as very likely to be updated soon.

Similarly to the two previous chapters, 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 hot 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.

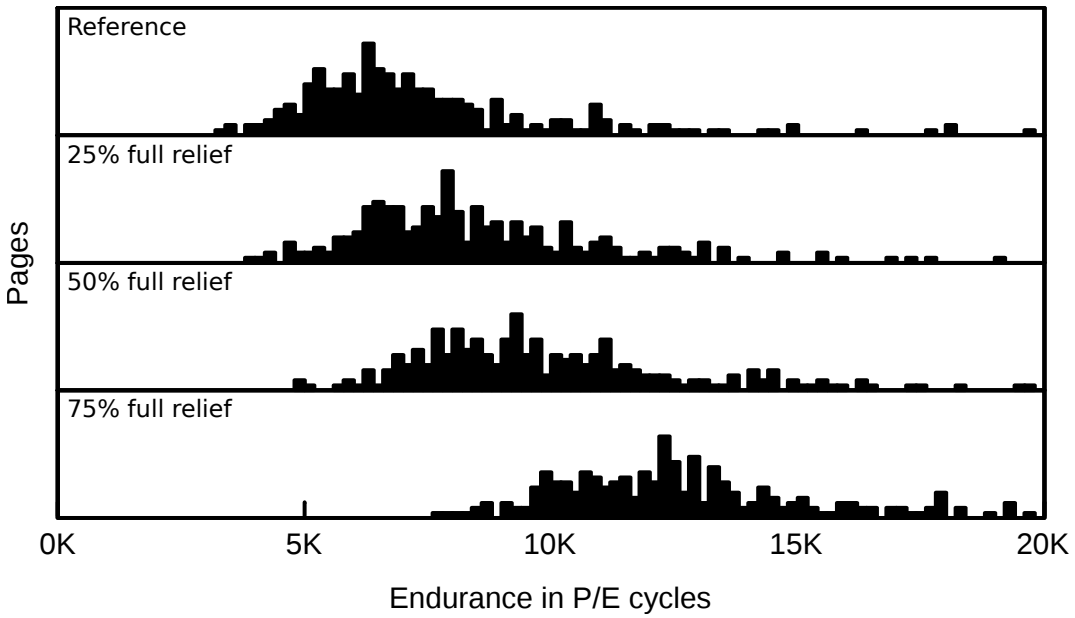


Figure 6.6: **Measured page endurance distribution.** For a given BER threshold, we report the measured page endurance distribution for every page sets of relief rates and compare them with the reference page endurance distribution. The clusters on the left and right correspond to MSB and LSB pages, respectively. Both clusters endurance are extended homogeneously when relieved.

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 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.

6.3.3 Reactive Approach: Identify 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 [2]. 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

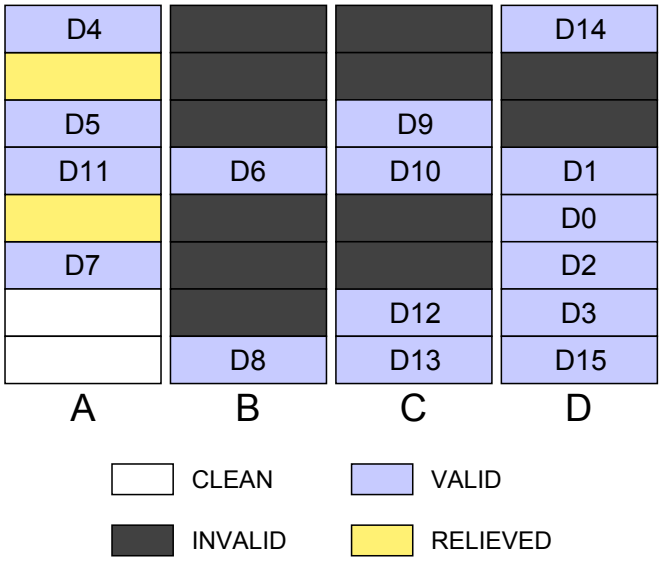


Figure 6.7: **Pages state transitions.** The figure illustrates the various page states found in a typical flash storage: *clean* when it has been freshly erased, *valid* when it holds valid data, and *invalid* when its data has been updated elsewhere. In this work, we opportunistically relieve weak pages to limit their cumulative stress. Accordingly, we introduce a fourth state, *relieved*, identifying pages currently relieved.

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 (remember that relief can be administered only to pages in the hot partitions). 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 as 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 contain-

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

6.3.4 Proactive Approach: Relieving Plan Ahead of Time

The *reactive* approach requires identifying 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 [22, 6]. 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.

	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 6.8: **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.

However, in practice, one cannot have this information ahead of time. Instead, we prepare a sequence of plans targeting increasing hot allocation counts; Figure 6.8 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 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 cycles 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

$$E_{X,i}(\rho_i, \alpha) = \frac{E_i}{(1-\rho_i) + \rho_i \alpha} = E_T \tag{6.4}$$

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

$$R_i = \frac{E_T - E_i}{1-\alpha}. \tag{6.5}$$

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 6.2.3, 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 (6.5) 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 (6.5) and with $L_p = R_i$, we determine the fraction λ of full relief cycles such that the average fraction of stress is

$$\alpha = \lambda \alpha_F + (1-\lambda)\alpha_H = 1 - \frac{E_T - E_i}{L_p}. \tag{6.6}$$

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 6.8, 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 6.8 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 reducing 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 storing 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 comparable to a page-level mapping table of a single block and is hence 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.

6.4 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 executing a large set of benchmark on our trace-driven simulator not only to show the lifetime improvement but also the minimal effect (often favorable) of our technique on execution time.

6.4.1 Collecting Real 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 behavior 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.

6.4.2 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, Figures 6.9 and 6.10 show 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 from the lowest lifetime on the left up to the the largest on the right. The bottom curve is the lifetime of each block when stressed normally, while the two curves on the top correspond to the lifetime when applying our techniques. The relief effectiveness varies depending on the actual block; 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.

We now evaluate the lifetime extension for a set of blocks when relieving the weakest pages. The three gray areas of Figures 6.9 and 6.10 represent the total amount of data we could write in 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: in the figure, 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 our technique and solutions proposed by Pan et al. [50], and Woo and Kim [57]. Essentially, the authors acknowledge the block endurance variance and designed techniques to try exploit to exploit a maximum the endurance of each individual block. If we were to predict the endurance of every block perfectly, we would have a device lifetime that is equal to the sum of every block lifetime, 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.

6.4.3 Device Lifetime Extension

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

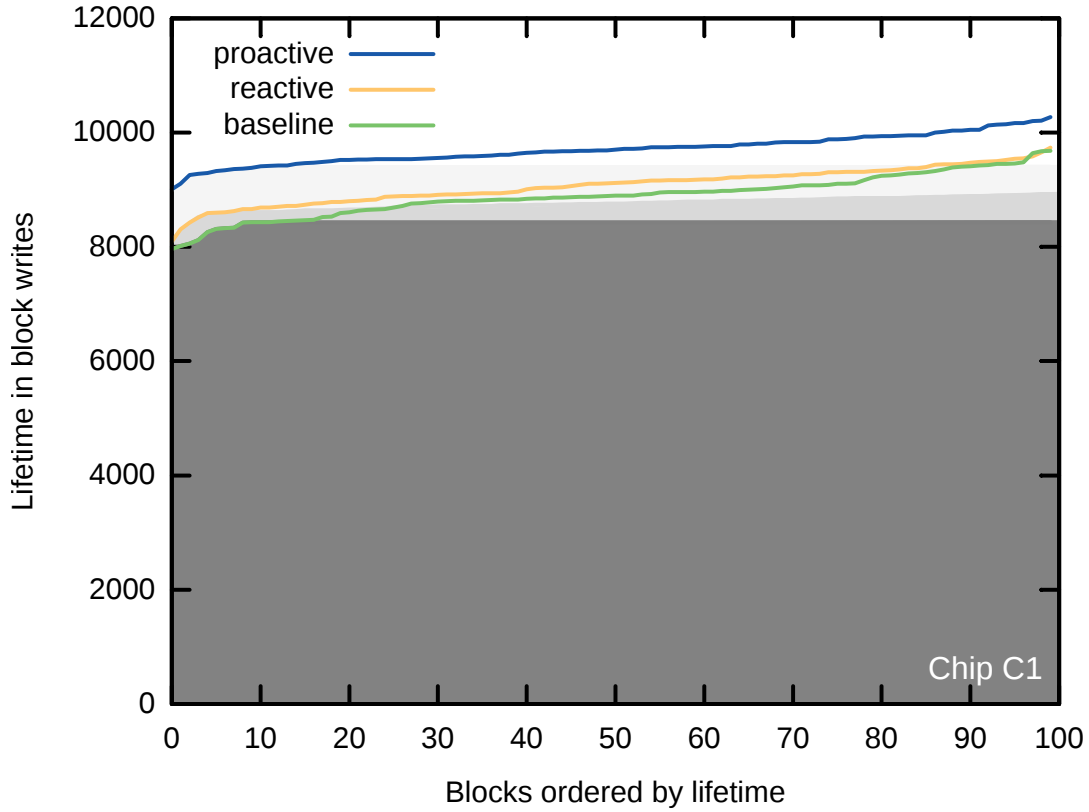


Figure 6.9: **Block lifetime improvement for C1.** The graphs shows for each chip how relieving weak pages can improve the device lifetime compared to a regular case. The curves show the individual block lifetime, and the surface areas the device lifetime, assuming it can accumulate up to 10% bad blocks. 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. For these graphs, we assume a limit BER of 10^{-4} as well as a 60% write frequency to the hot partition.

the BER threshold, the effect on lifetime extension is moderate, as illustrated in Figure 6.11. 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 6.12 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

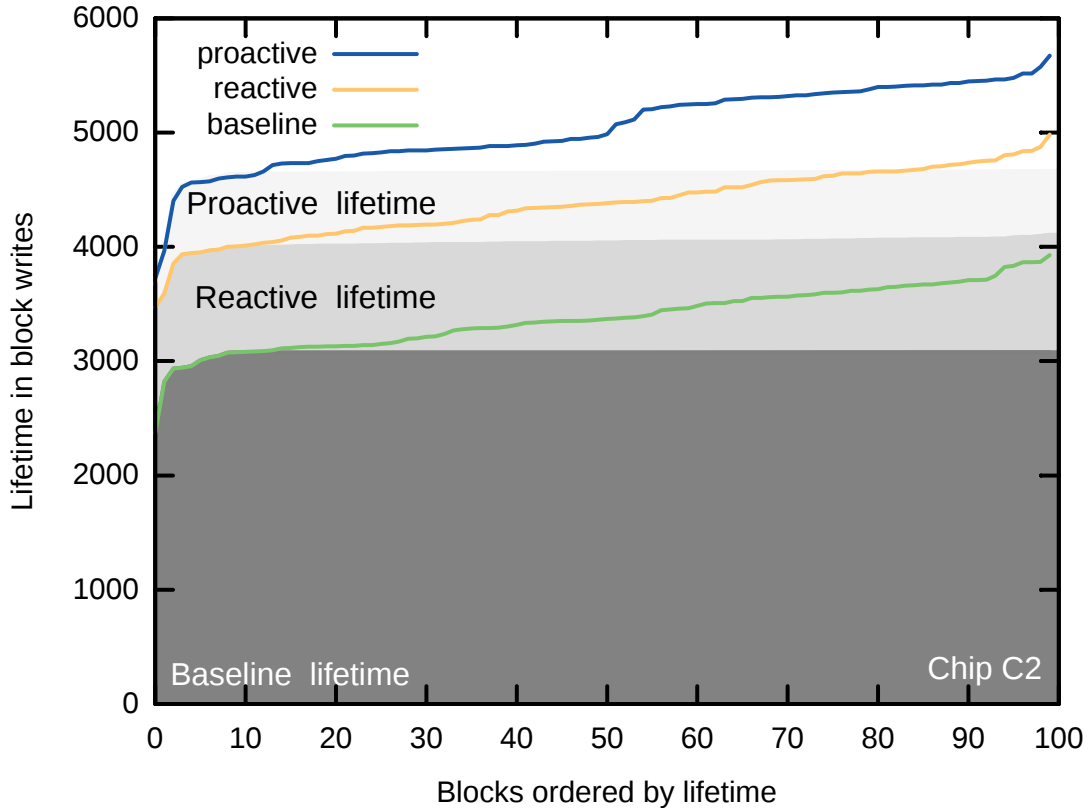


Figure 6.10: **Block lifetime improvement for C2.** With a larger endurance variance, C2 offers more room to exploit the relief mechanism than C1 and allows the *proactive* approach to extend by 50% the lifetime.

ratios larger than 40%, which is in fact in the range where most benchmarks (with very rare exceptions) fall in practice.

6.4.4 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 implement the relief strategy for ComboFTL and ROSE, two hybrid FTLs introduced in Chapter 4 and with their generic mapping architecture reminded in Figure 6.13. On the figure, we illustrate a weak page being relieved in the log buffer, whose capacity gets reduced accordingly. Compared to ROSE, ComboFTL has an extra 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

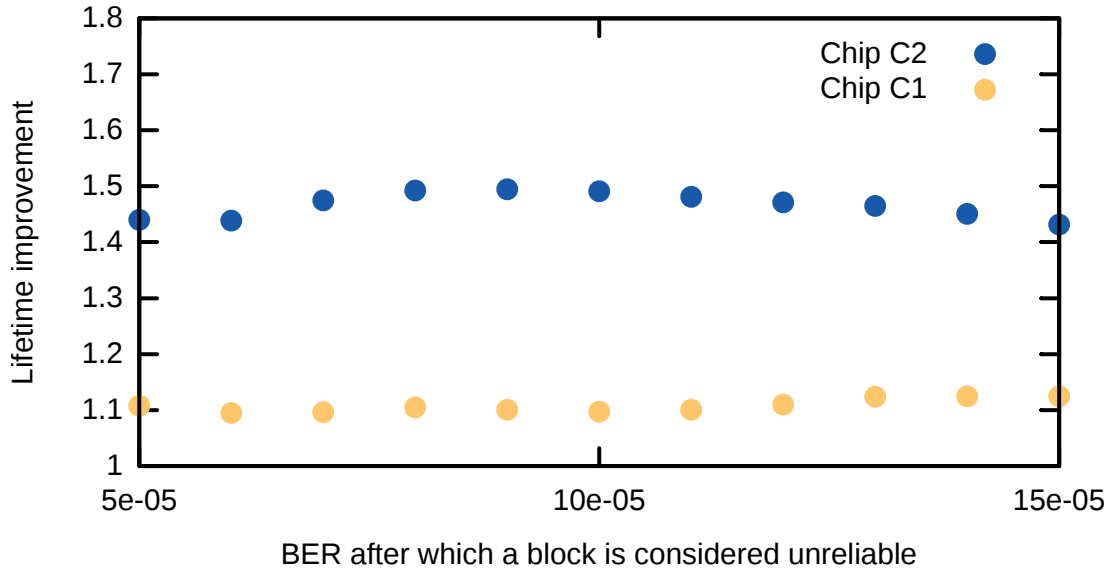


Figure 6.11: **Lifetime improvement w.r.t. BER threshold.** The BER threshold that indicates when a block is considered unreliable directly affects a device lifetime. Large BER thresholds increase the baseline lifetime and remove room to improvement at the cost of a more expensive ECC.

relief cycles when appropriate. Thanks to the block level mapping of its data partition, 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 be considered either as extra capacity requirement (1.25% here) or in a garbage collection overhead that we will now evaluate for two different FTLs.

We selected the large set of disk traces introduced in Chapter 4 to be executed by both FTLs. In our simulation, we assume again a total capacity of 16 GB and a flash device with the characteristics of C2 (see Table 6.1). As discussed in Section 4.5, when simulating a device smaller than the original trace source, 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 Figures 6.14 and 6.15 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, i.e., implementing the same FTL without relieving weak pages. (Note that this makes the results for ComboFTL and ROSE not comparable be-

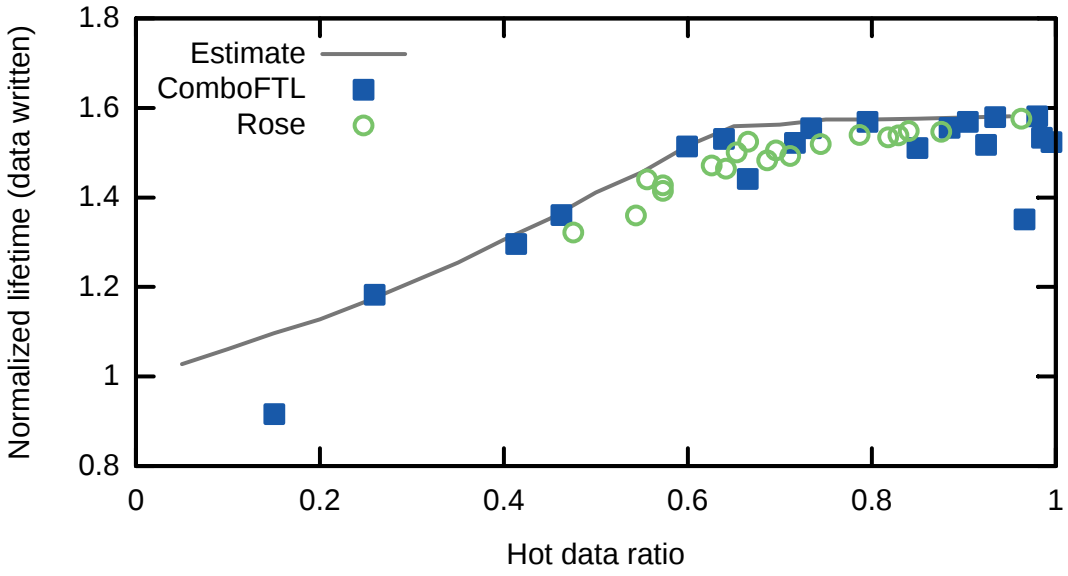


Figure 6.12: **Lifetime improvement w.r.t. hot write ratio.** The curve gives the expected lifetime extension provided by the *proactive* technique on chip C2. The data points represent results from benchmarks using two different FTLs. Those measurements take into account the writes overhead caused by the hot partition capacity loss. Apart from a couple of outliers, the results are consistent with our expectations.

tween themselves, but our purpose here is not to compare different FTLs but rather to show that, irrespective 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.

6.5 Related Work

Lue et al. suggest adding a built-in local heater on the flash circuitry [40], which would heat cells at 800C 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 of 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.,

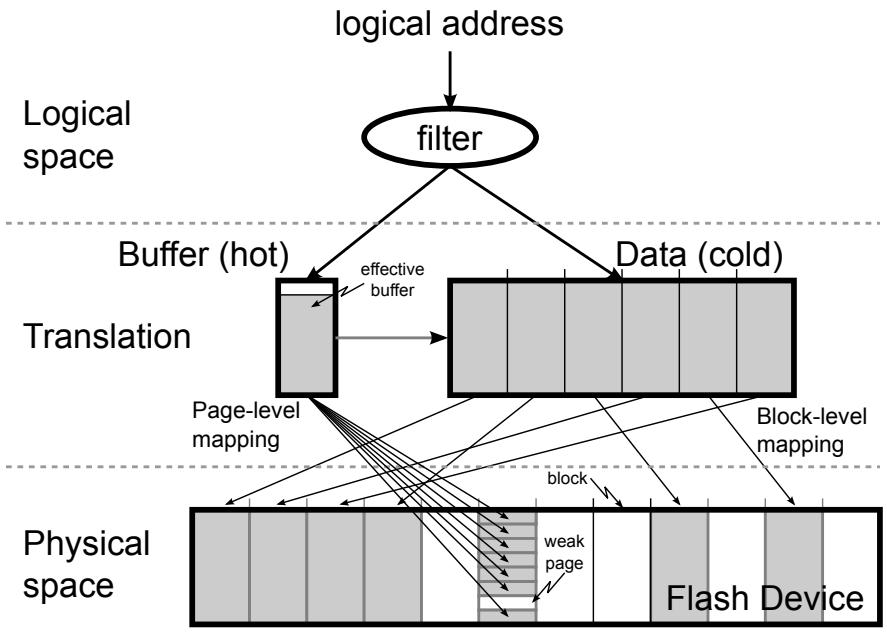
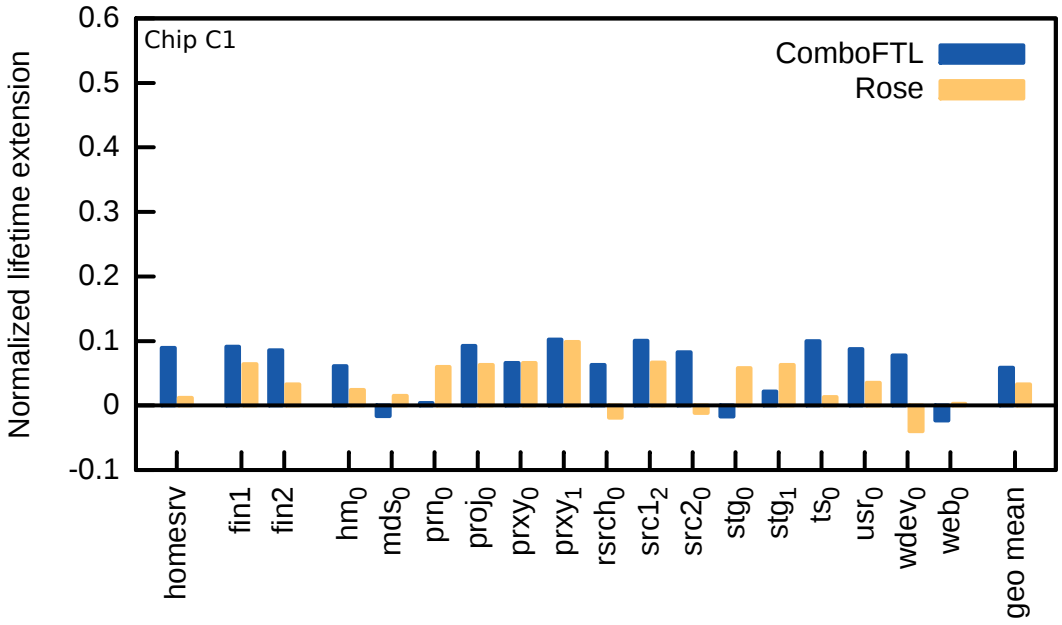


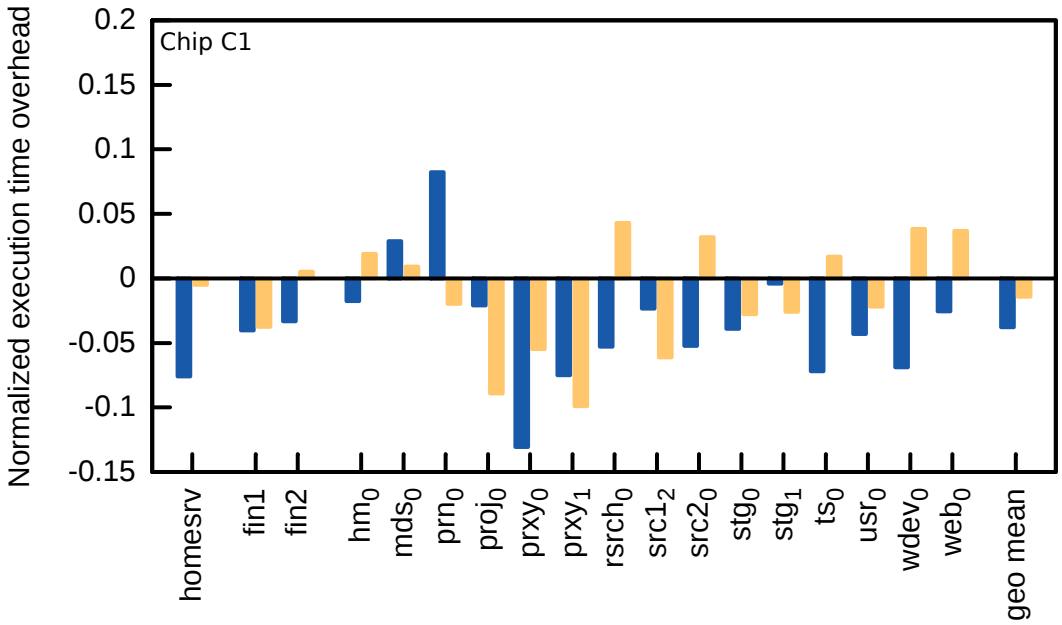
Figure 6.13: **Hybrid FTL.** A hybrid-FTL is composed of a large block-mapped partition, representing the device total logical capacity, and a small partition, buffering small updates. The buffer partition uses page-level mapping and helps to reduce the large garbage collection overhead inherent to a block-level mapping. The larger the buffer the smaller the overhead. Relieving pages from the buffer reduces its effective size and has a direct negative impact on performance.

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.

Pan et al. acknowledge the block endurance variance and suggest to adapt classical wear-leveling algorithms to compare blocks on their BER rather than their P/E cycles count [50]. 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. In a similar fashion, Woo and Kim propose to perform the wear-leveling on a new wear index that relies on the error count and the program latency [57]. They use the correlation between wear and program latency to refine their block wear estimate. However, in practice, the program latency cannot be distinguished when multiple commands are processed on the same channel, restricting the use of this technique to the simplest architectures. Lastly, as seen in Section 6.4.3, we observed significantly more potential acting at the page level rather than the block level.

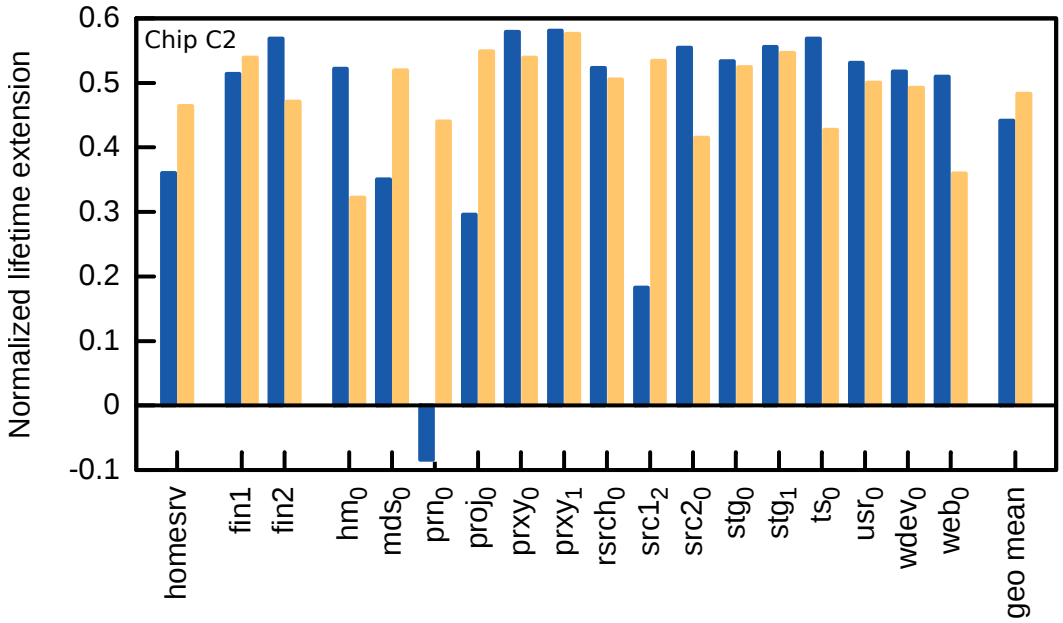


(a)

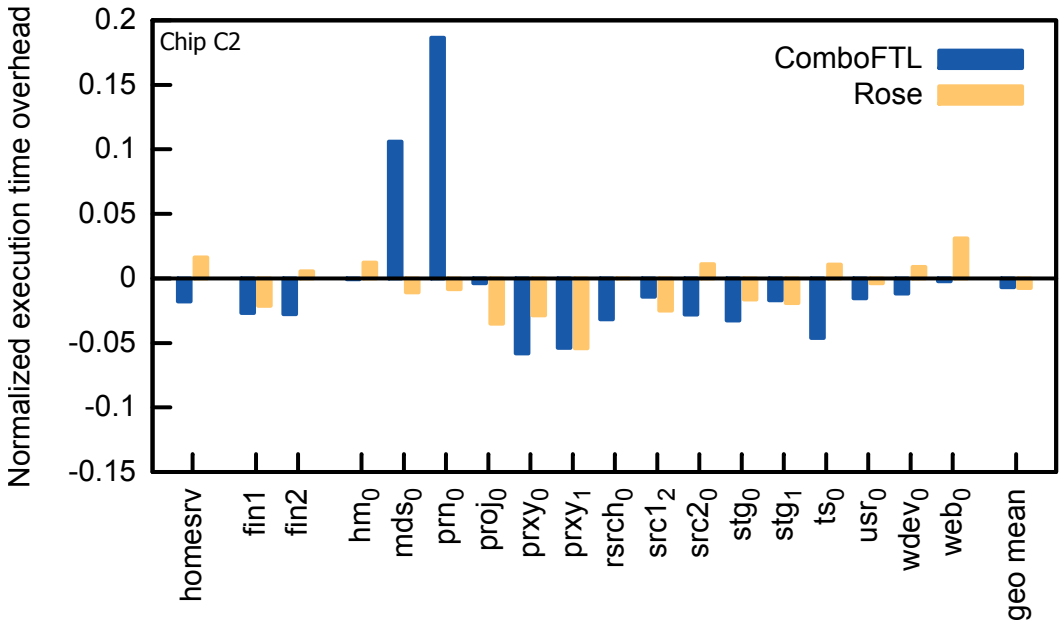


(b)

Figure 6.14: **Performance and lifetime evaluation of our *proactive* technique for various benchmarks running on chip C1.** (a) Our relief technique gets at most 10% lifetime extension for the chip C1. In (b), 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.



(a)



(b)

Figure 6.15: **Performance and lifetime evaluation of our *proactive* technique for various benchmarks running on chip C2.** (a) The *proactive* strategy is significantly more efficient for C2 than C1, with regularly 50% extra lifetime, but for rare exceptions. In (b), similarly than for C2, the execution time stays stable and better sometimes.

6.6 Conclusion

In this chapter, 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 balance the endurance of every page and block further. 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.

7 Conclusions

NAND flash memory is a mature technology that is facing many hurdles in the quest for higher densities. Despite those challenges and the fact that in theory many emerging memories promise better scalability, NAND flash remains the most viable semiconductor memory storage in the near future. Consequently, tremendous efforts are put into further developments of this technology (e.g., going to 3D NAND [25]), which necessarily will bring new types of NAND architectures with a new set of physical properties to investigate. We are convinced that exploiting new peculiar properties, sometimes (nearly) for free, will influence and complement the advances at the device level to further extend the possibilities and range of applications for NAND flash. In this thesis, we have shown that existing flash memories already exhibit neglected properties that can be used to improve their capabilities. In particular, we have presented several techniques exploiting these physical properties to enable sizable lifetime extensions in today's commercial NAND flash memories.

First, we were able to quantify the radically different wear occurring when MLC is used in SLC-mode. This allowed us to demonstrate that using alternately SLC-mode with MLC does not reduce the amount of data writable during the lifetime of an MLC chip. Interestingly, for the experimented chips, it even improves it slightly. Therefore, rather than having to build complex architecture made of heterogeneous memory to provide two classes of performance, we suggest to use only MLC chips in a flexible manner to address both classes of performance on a completely homogeneous architecture, which is simpler to control and will provide a larger lifetime than previously anticipated.

Second, we characterized the block endurance statistical variance on a NAND flash chip and, accordingly, proposed a model to describe the device degradation. The observation of this variance inspired us Phœnix, a strategy that is nearly free to implement and extends flash lifetime by exploiting the endurance variance through bad block revival. Combined with the device degradation model, we were able to qualify the potential of Phœnix and describe how the chip degradation parameters (i.e., endurance average and variance) would influence this potential.

Lastly, we characterized the page endurance within a block and observed a significant variance. Accordingly, we proposed to unbalance the wear within a block in order to level the page endurance by relieving weak pages. The effect of relieving a page is a neglected physical property that requires a particular care to be characterized accurately. We proposed two strategies to exploit page relief that require minimal resources to be implemented and can extend by up to 60% the flash lifetime, which is by far the highest improvement among other strategies addressing the endurance variance that we know of.

Overall, although the extracted characterized properties may vary quantitatively from one flash chip to another, each time we presented the methodology to extract them and proposed accurate models describing their effects. Based on the understanding of such mechanisms we designed original methods that help increasing flash devices lifetime while requiring very limited extra resources and being compatible with most exiting FTLs. Indeed, we strived to make these solutions as orthogonal as possible to traditional FTL policies and mapping algorithms to remove any risk of degrading other FTL objectives, such as performance.

A recurrent characteristic that we exploited in this thesis is the statistical variability of a physical property, such as the endurance or the performance. Understanding how this property correlates with other factors provides a better control on the device and gives opportunities to improve the overall device performance. Furthermore, for this thesis, we restricted the characterization granularity to the page level or larger. Yet, with flash pages progressively growing for each new technology node, designing techniques on smaller parts such as a sector level might open new perspectives. Therefore, we believe there is still a wide scope for research in that direction. We hope the light shed in this thesis on previously neglected properties inspires a revitalization of flash management strategies.

Bibliography

[1] David G. Andersen and Steven Swanson. Rethinking flash in the data center. *IEEE Micro*, 30(4):52–54, July 2010.

[2] Daniel L. Auclair, Jeffrey Craig, Daniel C. Guterman, John S. Mangan, Sanjay Mehrotra, and Robert D. Norman. Soft errors handling in EEPROM devices. US Patent 08/406,667, August 1997.

[3] Ken Bates and Bruce McNutt. OLTP application I/O. http://traces.cs.umass.edu/index.php/Storage/Storage, June 2007.

[4] Simona Boboila and Peter Desnoyers. Write endurance in flash drives: Measurements and analysis. In *Proc. USENIX Conf. on File and Storage Technologies*, San Jose, California, USA, February 2010.

[5] Joe Brewer and Manzur Gill. *Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices*. John Wiley & Sons, 2008.

[6] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 521–26, Dresden, Germany, March 2012.

[7] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 1285–90, Grenoble, France, March 2013.

[8] Paulo Cappelletti, Carla Golla, Piero Olivo, and Enrico Zanoni. *Flash Memories*. Kluwer Academic Publishers, 1999.

[9] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Proc. Int. Conf. Arch. Support for Programming Languages and Operating Systems*, pages 217–28, Washington, DC, USA, March 2009.

[10] Li-Pin Chang. Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs. In *Asia and South Pacific Design Automation Conf.*, pages 428–33, Seoul, Korea, January 2008.

[11] Li-Pin Chang. A hybrid approach to NAND-flash-based solid-state disks. *IEEE Trans. Computers*, 59(10):1337–49, October 2010.

[12] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. Endurance enhancement of flash-memory storage, systems: An efficient static wear leveling design. In *Design Automation Conf.*, pages 212–17, San Diego, California, USA, June 2007.

[13] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proc. Int. Joint Conf. on Measurement and Modeling of Computer Systems*, pages 181–92, Seattle, WA, USA, June 2009.

[14] Feng Chen, Tian Luo, and Xiaodong Zhang. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proc. USENIX Conf. File and Stroage Technologies*, San Jose, California, USA, February 2011.

[15] Mong-Ling Chiao and Da-Wei Chang. ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation. *IEEE Trans. Computers*, 60(6):753–66, June 2011.

[16] Hyunjin Cho, Dongkun Shin, and Young Ik Eom. KAST: K-associative sector translation for NAND flash memory in real-time systems. In *Design Automation and Test in Europe*, pages 507–12, Nice, France, April 2009.

[17] Peter Desnoyers. Empirical evaluation of NAND flash memory performance. In *Workshop on Hot Topics in Storage and File Systems*, Big Sky, Montana, USA, October 2009.

[18] Peter Desnoyers. What systems researchers need to know about NAND flash. In *Proc. USENIX Conf. Hot Topics in Storage and File Systems*, San Jose, California, USA, June 2013.

[19] Claudio Favi, René Beuchat, Xavier Jimenez, and Paolo Ienne. From gates to multi-processors learning systems hands-on with FPGA4U in a computer science programme. In *Proc. Workshop on Embedded Systems Education*, pages 56–63, Grenoble, France, October 2009.

[20] Ryan Gabrys and Lara Dolecek. Characterizing capacity achieving write once memory codes for multilevel flash memories. In *IEEE Int. Symp. Inf. Theory*, pages 2517–21, July 2011.

[21] Ryan Gabrys, Eitan Yaakobi, Lara Dolecek, Paul H. Siegel, Alexander Vardy, and Jack K. Wolf. Non-binary WOM-codes for multilevel flash memories. In *IEEE Inf. Theory Workshop*, pages 40–44, October 2011.

[22] Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *ACM/IEEE Int. Symp. Microarchitecture*, pages 24–33, New York, NY, USA, December 2009.

[23] Laura M. Grupp, John D. Davis, and Steven Swanson. The bleak future of NAND flash memory. In *USENIX conf. on File and Storage Technologies*, San Jose, California, USA, February 2012.

[24] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proc. Int. Conf. Arch. Support for Programming Languages and Operating Systems*, pages 229–40, Washington, DC, USA, March 2009.

[25] Yi-Hsuan Hsiao, Hang-Ting Lue, Tzu-Hsuan Hsu, Kuang-Yeu Hsieh, and Chih-Yuan Lu. A critical examination of 3D stackable NAND flash memory architectures by simulation study of the scaling capability. In *IEEE International Memory Workshop*, pages 1–4, May 2010.

[26] Soojun Im and Dongkun Shin. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *Journal of Systems Architecture*, 56(12):641–53, December 2010.

[27] Anxiao Jiang and Jehoshua Bruck. Joint coding for flash memory storage. In *IEEE International Symposium on Inf. Theory*, pages 1741–45, July 2008.

[28] Xavier Jimenez, David Novo, and Paolo Ienne. Software controlled cell bit-density to improve NAND flash lifetime. In *Design Automation Conf.*, pages 229–34, San Francisco, California, USA, June 2012.

[29] Xavier Jimenez, David Novo, and Paolo Ienne. Phœnix: Reviving MLC blocks as SLC to extend NAND flash devices lifetime. In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 226–29, Grenoble, France, March 2013.

[30] Xavier Jimenez, David Novo, and Paolo Ienne. Libra: Software controlled cell bit-density to balance wear in NAND flash. *In press for ACM Trans. on Embedded Computing*, 2014.

[31] Xavier Jimenez, David Novo, and Paolo Ienne. Wear unleveling: Improving NAND flash lifetime by balancing page endurance. In *Proc. USENIX Conf. File and Storage Technologies*, pages 47–59, Santa Clara, California, USA, February 2014.

[32] Yongsoo Joo, Youngjin Cho, Donghwa Shin, Jaehyun Park, and Naehyuck Chang. An energy characterization platform for memory devices and energy-aware data compression for multilevel-cell flash memory. *ACM Trans. Design Automation of Electronic Systems*, 13(3):43:1–29, July 2008.

[33] Scott Kayser, Eitan Yaakobi, Paul H. Siegel, Alexander Vardy, and Jack K. Wolf. Multiple-write WOM-codes. In *Allerton Conf. Communication, Control, and Computing*, pages 1062–68, September 2010.

[34] Taeho Kgil and Trevor Mudge. FlashCache: A NAND flash memory file cache for low power web servers. In *Proc. Int. Conf. Compilers, Arch. and Synth. Emb. Sys.*, pages 103–12, Seoul, Korea, October 2006.

[35] Taeho Kgil, David Roberts, and Trevor Mudge. Improving NAND flash based disk caches. In *Proc. Int. Symp. Computer Architecture*, pages 327–38, Beijing, China, 2008. IEEE Computer Society.

[36] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for CompactFlash systems. *IEEE Trans. Consumer Electronics*, 48(2):366–75, May 2002.

[37] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embedded Computing Systems*, 6(3), July 2007.

[38] Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim. FlexFS: a flexible flash file system for MLC NAND flash memory. In *USENIX Annual Technical Conf.*, San Diego, California, USA, June 2009.

[39] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. *ACM SIGOPS Operating Systems Review*, 42(6):36–42, October 2008.

[40] Hang-Ting Lue, Pei-Ying Du, Chih-Ping Chen, Wei-Chen Chen, Chih-Chang Hsieh, Yi-Hsuan Hsiao, Yen-Hao Shih, and Chih-Yuan Lu. Radically extending the cycling endurance of flash memory (to >100M cycles) by using built-in thermal annealing to self-heal the stress-induced damage. In *IEEE Int. Electron Devices Meeting*, pages 9.1.1–4, San Francisco, California, USA, December 2012.

[41] Dongzhe Ma, Jianhua Feng, and Guoliang Li. LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *Proc. SIGMOD Int. Conf. on Management of Data*, Athens, Greece, June 2011.

[42] Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Flouris, and Angelos Bilas. Using transparent compression to improve SSD-based I/O caches. In *Proc. European Conf. on Computer Systems*, EuroSys '10, pages 1–14, Paris, France, April 2010.

[43] Rino Micheloni, Luca Crippa, and Alessia Marelli. *Inside NAND Flash Memories*. Springer, 2010.

[44] Micron. Bad block management in NAND flash memory. http://www.micron.com/products/support/technical-notes/, October 2010.

[45] Neal Mielke, Hanmant P. Belgal, Albert Fazio, Qingru Meng, and Nick Righos. Recovery effects in the distributed cycling of flash memories. In *IEEE Int. Reliability Physics Symp. Proc.*, pages 29–35, San Jose, California, USA, March 2006.

[46] Vidyabhushan Mohan, Taniya Siddiqua, Sudhanva Gurumurthi, and Mircea R. Stan. 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.

[47] Muthukumar Murugan and David H.C. Du. Hybrot: Towards improved performance in hybrid SLC-MLC devices. In *IEEE Int. Symp. Modeling, Analysis Simulation of Computer and Telecommunication Systems*, pages 481–84, Arlington, Virginia, USA, August 2012.

[48] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. In *Proc. USENIX Conf. File and Storage Technologies*, pages 253–67, San Jose, California, USA, February 2008.

[49] ONFI. *Open NAND Flash Interface 3.2*, June 2013.

[50] Yangyang Pan, Guiqiang Dong, and Tong Zhang. Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes. *IEEE Trans. Very Large Scale Integration Systems*, 21(7):1350–54, July 2013.

[51] Dongchul Park, Biplob Debnath, Youngjin Nam, David H. C. Du, Youngkyun Kim, and Youngchul Kim. HotDataTrap: a sampling-based hot data identification scheme for flash memory. In *ACM Int. Symp. Applied Computing*, pages 1610–17, Riva del Garda, Italy, March 2012.

[52] Jung-Wook Park, Seung-Ho Park, Charles C. Weems, and Shin-Dug Kim. A hybrid flash translation layer design for SLC-MLC flash memory based multibank solid state disk. *Microprocessors & Microsystems*, 35(1):48–59, February 2011.

[53] Timothy Pritchett and Mithuna Thottethodi. SieveStore: A highly-selective, ensemble-level disk cache for cost-performance. In *Proc. Int. Symp. on Computer Architecture*, pages 163–74, Saint-Malo, France, June 2010.

[54] Frankie Roohparvar. Single level cell programming in a multiple level cell non-volatile memory device. US Patent 11/298,013, June 2007.

[55] Mohit Saxena, Michael M. Swift, and Yiying Zhang. FlashTier: A lightweight, consistent and durable storage cache. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 267–80, Bern, Switzerland, 2012. ACM.

[56] Chundong Wang and Weng-Fai Wong. Extending the lifetime of NAND flash memory by salvaging bad blocks. In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 260–63, Dresden, Germany, March 2012.

[57] Yeong-Jae Woo and Jin-Soo Kim. Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In *Proc. Int. Conf. Embedded Software*, pages 1–10, Montreal, Canada, September 2013.

[58] David Woodhouse. JFFS : The journalling flash file system. In *Proc. Linux Symp.*, Ottawa, Ontario, Canada, July 2001.

[59] Guanying Wu and Xubin He. Delta-FTL: Improving SSD lifetime via exploiting content locality. In *Proc. European Conf. Computer Systems*, pages 253–66, Bern, Switzerland, April 2012.

[60] Guanying Wu, Xubin He, Ningde Xie, and Tong Zhang. Exploiting workload dynamics to improve SSD read latency via differentiated error correction codes. *ACM Trans. Des. Autom. Electron. Syst.*, 18(4):55:1–22, October 2013.

[61] Michael Wu and Willy Zwaenepoel. eNVy: a non-volatile, main memory storage system. In *Sixth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 86–97, San Jose, California, USA, October 1994.

Xavier Jimenez

Computer Science PhD

Rte Neuve 1
1041 Dommartin
✉ xav.jimenez@gmail.com
Born 31th July 1983, Swiss
Married with 2 children

Education

2008–2014 **Ph.D. in Computer Science**, *Ecole Polytechnique Fédérale de Lausanne*.

2002–2007 **Master in Computer Science**, *Ecole Polytechnique Fédérale de Lausanne*.
Master Thesis: Design and Implementation of a 5-stage Pipeline Nios II

Bibliography

2014 Xavier Jimenez, David Novo, and Paolo Ienne. **Libra: Software controlled cell bit-density to balance wear in NAND flash.** In *press for ACM Trans. on Embedded Computing*

February 2014 Xavier Jimenez, David Novo, and Paolo Ienne. **Wear unleveling: Improving NAND flash lifetime by balancing page endurance.** In *Proc. USENIX Conf. File and Storage Technologies*, pages 47–59, Santa Clara, California, USA

March 2013 Xavier Jimenez, David Novo, and Paolo Ienne. **Phoenix: Reviving MLC blocks as SLC to extend NAND flash devices lifetime.** In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 226–29, Grenoble, France

June 2012 Xavier Jimenez, David Novo, and Paolo Ienne. **Software controlled cell bit-density to improve NAND flash lifetime.** In *Design Automation Conf.*, pages 229–34, San Francisco, California, USA

October 2009 Claudio Favi, René Beuchat, Xavier Jimenez, and Paolo Ienne. **From gates to multiprocessors learning systems hands-on with FPGA4U in a computer science programme.** In *Proc. Workshop on Embedded Systems Education*, pages 56–63, Grenoble, France

Experience

Since 2013 **R&D Project Manager**, *senseFly*, Cheseaux-sur-Lausanne.
- Managed various projects of camera integration for light fixed-wing UAVs;
- Leaded the development of an ultra-light multispectral camera for precision agriculture.

2008–2014 **Ph.D.**, *Processor Architecture Laboratory (LAP) EPFL*, Lausanne.
- Created new techniques to improve the management of consumer NAND flash memories;
- Modeled and evaluated them through experimental characterization and benchmarking;
- Applying concepts from computer science, physics and mathematics;

2007–2013 **Research Assistant**, *Processor Architecture Laboratory (LAP) EPFL*, Lausanne.

- Managed for three years the FPGA4U project (`http://fpga4u.epfl.ch`), a hardware platform for Computer Science and Communication Systems students;
- Coordinated various processor implementations in VHDL for FPGA (ARM, Nios II, OpenRISC);
- Taught Processor Architecture, Digital Design and Real Time Embedded Systems.

2012 **Hardware Engineering**, *armasuisse*, Bern.

- Designed and implemented a hardware accelerator for a large parallel iterative function;
- Enabled a 3'000× speed-up on FPGA compared to the software reference implementation.

2006 **Hardware Engineering**, *Processor Architecture Laboratory (LAP) EPFL*, Lausanne.

- Created a complete test bench for the FPGA4U boards (`http://fpga4u.epfl.ch`);
- Detected and identified successfully hundreds defects with 0% false positive.

Skills

Programming	Java, C, C++, VHDL, Verilog, Python, Assembly, PHP, SQL, XML, Matlab, UML
Tools	Unix utilities, Altera Quartus and Qsys, Xilinx ISE and EDK, Eclipse
OS	Linux, Unix, Windows

Languages

French	First language
English	Fluent
Spanish	Conversational