ARCHITECTURAL SUPPORT FOR DESIGNING DEPENDABLE NON-VOLATILE MAIN MEMORIES

A Dissertation Presented
by
Mohammad Khavari Tavana
to
The Department of Electrical and Computer Engineering
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

March 2018
To my loving wife, and compassionate parents
# Contents

**List of Figures**

List of Tables vii

List of Acronyms viii

Acknowledgments x

Abstract xi

1 Introduction 1

1.0.1 Summary of Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.0.2 Organization of the thesis . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 Background 6

2.1 PCM Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 PCM Fault Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.3 PCM-based Memory Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.4 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.4.1 Prior Work: Hard-Fault Tolerance in PCM . . . . . . . . . . . . . . . . . . . 11
2.4.2 Prior Work: Write Disturbance . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.4.3 Prior Work: Secure Non-volatile Memory . . . . . . . . . . . . . . . . . . . 13

3 REMAP: A Reliability/Endurance Mechanism for Advancing PCM 16

3.1 REMAP: A Reliability/Endurance Mechanism for Advancing PCM . . . . . . . . 17
3.1.1 Example: Error Correction with REMAP in Read/Write Operation . . . . 19
3.1.2 Experimental Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.2 Extra Write Overhead Alleviation . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.2.1 Write Overhead Localization Using Partitioning . . . . . . . . . . . . . . . 24
3.2.2 Fault Location Caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.2.3 A Second Layer of Defense . . . . . . . . . . . . . . . . . . . . . . . . . . 29
3.2.4 Comparing All Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . 31

3.3 Trace-driven lifetime evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

3.3.1 Experimental Setup for Trace-driven Simulation . . . . . . . . . . . . . . . 32
List of Figures

2.1 A PCM cell (a), and read/write operations in PCM (b). .............................................. 7
2.2 PCM cell when it is (a) SET-stuck failure (SSF), and (b) RESET-stuck failure (RSF). 8
2.3 The baseline architecture of a PCM-based DIMM. ...................................................... 9
3.1 Write operation to a faulty memory block. ............................................................... 18
3.2 Read operation from a faulty memory block. ............................................................ 20
3.3 Page lifetimes for various error-correction schemes, assuming a range of variances:  
   (a) 0.20, (b) 0.25, and (c) 0.30. .............................................................. 21
3.4 Probability of having a different number of faults in data blocks with CV=0.25 for:  
   (a) REMAP and (b) Aegis 23×23 schemes. ......................................................... 22
3.5 Partitioning with minimum coverage. If (a) slope = 0 is employed, four partitions  
   (0, 1, 2 and 4) are required to cover the four existing faults within the block, but  
   when we select (b) with the slope = 1, the number of faulty partitions is reduced to  
   two (4 and 5). ................................................................. 26
3.6 Memory lifetimes for REMAP with static and dynamic partitioning, and with differ-  
   ent variances, (a) 0.20, and (b) 0.30. ............................................................... 28
3.7 Memory lifetime with different cache hit rates when using REMAP. .......................... 29
3.8 Lifetime enhancement when using REMAP as a second layer of defense. The  
   increased lifetime is added as a stacked bar on top of each scheme. ......................... 30
3.9 Page lifetimes after applying optimizations to REMAP, with CV values of (a) 0.20,  
   (b) 0.25, and (c) 0.30. ................................................................. 31
3.10 Lifetime across our workloads for different ECCs with CV= 0.20. The results are  
     normalized to the ECP6 scheme. ............................................................... 34
3.11 Lifetime across our workloads for different ECCs with CV= 0.30. The results are  
     normalized to the ECP6 scheme. ............................................................... 35
3.12 A hardware implementation for bit replacement. ...................................................... 36
3.13 Performance loss across our benchmarks, as compared to the baseline system, for  
     different error rates. The baseline is a memory system without any errors. ............. 38
4.1 Memory lifetime and the average error per page for different scenarios. Each scenario  
     mimics different behavior of workloads or memory system. Non-uniformity in write  
     accesses reduces PCM memory lifetime significantly and results in under-utilization  
     of error correcting chip for error recovery. ......................................................... 42
4.2 Block cooperation statechart. At the highest level of the statechart hierarchy, the state of blocks is either **faulty** or **non-faulty**. Each of these superstates contains various states, which transitions to a new state based on the specific operation.

4.3 ECP metadata format when using (a) single level, and (b) multiple level, metadata sharing.

4.4 Integrating block cooperation in Aegis.

4.5 The format of the metadata for Aegis ($17 \times 31$) with block cooperation.

4.6 Lifetimes using standard ECP6, single level metadata sharing, and multiple level metadata sharing, with CoV values of (a) 0.20, and (b) 0.30.

4.7 Lifetimes using standard Aegis, Aegis with single sharing level, and Aegis with multiple sharing level, with CoV values of (a) 0.20, and (b) 0.30.

4.8 Simulation framework and methodology for workload characterization and endurance analysis. * symbol indicates the outputs of our simulation framework.

4.9 Distribution of (a) writes, and (b) bit flips at block level for different benchmarks. The red dot shows the median of the distribution.

4.10 Distribution of (a) writes, and (b) bit flips at a page level for different benchmarks. The red dot shows the median of the distribution.

4.11 Y axis (petabyte writes) shows the possible amount of writes before half of the memory capacity becomes unavailable. CoV is equal to 0.2.

4.12 Y axis (petabyte writes) shows the possible amount of writes before half of the memory capacity become unavailable. CoV is equal to 0.3.

4.13 Memory durability (as petabyte write) versus the average error per page for different error correction schemes. Using block cooperation increases the number of correctable errors per page and increases metadata utilization for both ECP and Aegis.

5.1 Writing a vulnerable data pattern in PCM. After a write, a '0' value in one of the exposed bits (cell#7) is changed to a '1'.

5.2 Writing to all the exposed bits to eliminate any write disturbance results in a domino write to all the neighboring idle '0' cells (green dashed square).

5.3 DIN is not effective when a benchmark shows low compressibility.

5.4 (a) Manipulator pattern selection, as well as data encoding for different partitions, which is performed on write accesses. Data is encoded before being written to the main memory to reduce the chance of consecutive zeros. (b) Decoding data per each partition when it is fetched into the chip. (c) Manipulator selection logic. (d) Encoder/Decoder logic.

5.5 DMPart effectively reduces the number of exposed bits in most benchmarks by modifying the written data.

5.6 Relationship among error rate, number of exposed bits and the write failure probability. When the number of exposed cells is high, it is not cost-effective to lower the write failure probability by writing to them.

5.7 Benchmark sensitivity to write disturbance. We evaluate the effectiveness of our proposed schemes, using benchmarks which have moderate to high sensitivity.

5.8 Expected value of writes. SWEX shows more robust performance, as compared to other schemes.

5.9 Breakdown of the energy consumption, normalized to the VnC scheme.
5.10 Bit flips across the different write disturbance mitigation techniques. Each result is normalized to VnC. DMPart uses 8B partitions, and \( \eta \) is set to 32 in SWEX.

5.11 Page lifetimes in writes-per-page, when different schemes are used for write disturbance mitigation.

6.1 Consecutive writes of an encrypted (a), and unencrypted (b) memory block for astar benchmark. Encryption increases the number of bit flips.

6.2 Lifetime reduction and energy increase for encrypted PCM memory vs. unencrypted PCM memory. The Y-axis on the left shows the lifetime reduction (solid bars), and Y-axis on the right shows the energy increase (dashed bars), normalized to the unencrypted memory.

6.3 (a) Target architecture, (b) PCM-based DIMM with comparison write capability, and (c) Counter-mode encryption. Page counters and line counters which are part of the IV is kept on the chip near the encryption engine.

6.4 A simple example which shows Nacre’s function at writebacks events step by step. SMI shows modified segments at each writeback. At each step one of the segment counters (SCs) is selected and the masks metadata are updated accordingly.

6.5 Impact of segment size on the number of bit flips. Nacre (Segment Size, Number of Counters).

6.6 Impact of number of segment counters on the number of bit flips. Nacre (Segment Size, Number of Counters).

6.7 Average number of modified bits per write with various encryption schemes. Unencrypted memory, Nacre, DEUCE, and BLE experience 11.84%, 17.46%, 26.27%, and 32.13% bit flips, respectively.

6.8 Memory energy consumption normalized to the unencrypted memory. Nacre only incurs 6% more energy on average.

6.9 Memory lifetime using various encryption schemes.
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Workload characteristics</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Main characteristics of the simulated system</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>The system specifications of our simulated system</td>
<td>53</td>
</tr>
<tr>
<td>4.2</td>
<td>Collected Metrics Related to Bit Flips at Block and Page level for Different Benchmarks</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Collected Metrics Related to Read/Write Accesses of Different Benchmarks</td>
<td>56</td>
</tr>
<tr>
<td>5.1</td>
<td>The system specifications of our simulated system</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>The system specifications of our modeled system</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Storage overhead for various schemes</td>
<td>95</td>
</tr>
</tbody>
</table>
List of Acronyms

PCM  Phase Change Memory.
ECP  Error Correction Pointer.
ECC  Error Correcting code.
SLC  Single-Level Cell.
MLC  Multi-Level Cell.
TLC  Triple-Level Cell.
SSF  SET-stuck failure.
RSF  RESET-stuck failure.
SECDED  Single Error Correction, Double Error Detection.
DIMM  Dual In-Line Memory Module.
AMB  Advanced-Memory Buffer.
RMW  Read Modify Write.
DRM  Dynamically Replicated Memory.
CV  Coefficient of Variation.
SLD  Second Layer of Defense.
LLC  Last Level Cache.
LLWB  Line Level Write-Back.
LSTP  Low Standby Power.
MSHR  Miss Status Holding Register.
FRAM  Ferroelectric RAM.
STT-MRAM  Spin Transfer Torque Memory RAM.
WD  Write Disturbance.
DIN  Data encoding-based INsulation.
BCH  Bose-Chaudhuri-Hocquenghem.
DMPart  Data Modification with Partitioning.
SWEX  Selective Write to the Exposed cells.
DCW  Data Comparison Write.
VnC  Verify and Correct.
WEX  Write to all the EXposed cells.
FPC  Frequent Pattern Compression.
NVM  Non-Volatile Memory.
BLE  Block Level Encryption.
AES  Advanced Encryption Standard.
OTP  One-Time Pad.
SC  Segment Counter.
CMP  Chip Multi-Processor.
SMI  Segment Modification Indicator.
Acknowledgments

I would like to dedicate this thesis to my parents and my spouse. Without their vast and boundless support I would not be a person that I stand today. I am especially indebted to my spouse Nasibeh for her love, patience, and encouragement during the hard times in the course of my Ph.D.

I am thankful to my Ph.D. committee members for providing me valuable inputs and enlighten me with new avenues and directions in my research. Particularly, I am thankful to Dr. Yunsi Fei for the insightful discussions in the area of security.

Finally, I would like to express my deepest gratitude to my advisor Dr. David Kaeli, the one whose with continuous support and guidance showed me the right path throughout my Ph.D. More importantly, his attitude and words of wisdom inspired me and changed my outlook toward life.

This work supported in part by NSF/SRC STARSS award #1618379.
Abstract

ARCHITECTURAL SUPPORT FOR DESIGNING DEPENDABLE NON-VOLATILE MAIN MEMORIES

by
Mohammad Khavari Tavana
Doctor of Philosophy in Computer Engineering
Northeastern University, March 2018
Dr. David Kaeli, Adviser

Rapid technology scaling has enabled the integration of many computational cores into a single chip. Given this level of core integration, the requirements for a large and scalable main memory system will only grow. Current DRAM-based main memory systems face power and scalability issues due to working at sub-micron scales. As existing memory technologies approach their physical limits in terms of scalability and power consumption, system designers have motivated to explore alternative technologies to meet the increasing demand for higher memory capacity. Phase Change Memory (PCM) has been proposed as one of the most promising technology candidates to replace or complement DRAM. PCM provides non-volatility, fast access latency, low power consumption, high scalability and bit-level access. However, dependability issues are the primary impediments toward adoption of PCM for next generation of memory systems.

In this thesis we investigate new and novel architectural support to design dependable non-volatile memories, while introducing minimal overhead. In the first part of the thesis we introduce mechanisms to improve cell wear-out in PCM technology, in order to prolong memory lifetime. PCM cells can only endure a limited number of writes, and will then wear out, therefore, PCM error correcting schemes are a must for reliable operation. Current error correction schemes for PCMs have limited capabilities in tolerating multiple errors. We first propose an error recovery mechanism to tolerate a large number of hard errors to improve reliability of PCMs. Our mechanism exploits metadata for replacing faulty bits error detection and location information is not needed. The location of failed memory cells are identified by a read verification, coupled with an extra write operation. Static and a dynamic partitioning schemes are proposed to alleviate the negative impacts of the extra writes, extending memory lifetime. Furthermore, we introduce a block-level cooperation technique that operates on top of error correction mechanisms to increase metadata utilization. Once an error
recovery scheme fails to recover from faults in a data block, the entire physical page associated with that block is disabled and becomes unavailable to the physical address space. To reduce the page waste caused by early block failures, other blocks can be used to support the failed block, working cooperatively to keep it alive and extend the faulty pages lifetime. We show how block cooperation can be realized through metadata sharing, and data layout reorganization besides state-of-the-art error correcting schemes.

In the second part of the thesis, we address thermal disturbance in very dense PCMs. Due to the heat generated during the programming of cells, neighboring cells may be disturbed, experiencing changes in their values. A naive solution is to increase inter-cell space, attempting to isolate cell programming and eliminating write disturbance, but this approach significantly reduces PCM density. We propose two cost-effective solutions to reduce the probability of write disturbance. Our solutions come with few side-effects on other memory system metrics. The first technique is based on data encoding, and tries to reduce the number of vulnerable data patterns when writing data to main memory. The second technique detects vulnerable cells, and overwrites them if their occurrence is below a set threshold. The proposed techniques are general and can avoid much of the performance overhead introduced by write disturbance with incurring negligible overhead on energy consumption and memory lifetime.

Lastly, this thesis addresses security as another aspect of dependability in the memory systems. While nonvolatility is a desirable feature to save energy in NVMs, it creates security vulnerabilities, since data will persistent after system power-off. Data stored in NVMs can be secured using data encryption. However, side effects imposed by memory encryption result in excessive bit writes, which will drastically reduce NVM lifetimes and increase energy consumption. We propose Nacre to bridge the gap between fully-encrypted and unencrypted NVMs. The Nacre exploits standard counter-mode encryption to maintain security. By tracking the different versions of the modified data in each memory writeback, the Nacre attempts to limit the number of bit writes. Selective reencryption is performed based on the history of the modified data in cache lines. We show that our security mechanism can improve memory lifetime significantly, with only marginal increases in energy consumption.
Chapter 1

Introduction

Modern applications execute many concurrent threads, producing a large memory footprint \[15\]. To sustain the inherent performance demands of these applications, next-generation systems will require large main memory systems that can scale to support this level of processing. These memory systems need to be reliable, while providing low latency and high bandwidth. This is particularly true for commercial servers, supercomputers and other large-scale shared environments.

DRAM has been the basic technology used for main memory systems during the past three decades. It can scale in terms of density, which has been a prime enabler for increasing main memory capacity. Due to shrinking cell sizes in successive technology nodes, DRAM scaling has become a challenging problem \[36\]. Due to limits on the electrical charge that can be trapped in DRAM capacitors, retention time is reduced, dramatically increasing the chance of unreliable sensing \[36, 46\]. This has motivated the architecture community to explore alternative memory technologies \[50, 67, 89, 95\].

Resistive memory technology has been shown to exhibit high potential as a DRAM replacement \[50, 95\] or DRAM complement \[24, 68\]. Instead of representing data values using the bit state, based on the presence or absence of electrical charge, resistive memories work by measuring the resistivity of cells. The cell resistivity is determined by the atomic arrangement of the memory material. Some examples of resistive memory technologies are Phase-Change Memory (PCM), Ferroelectric RAM (FRAM), Spin Transfer Torque Memory (STT-MRAM), and memristors. PCM is the most promising of these choices due to its maturity and the advances made in technology over the past decade. PCM has a number of desirable characteristics, including zero leakage power and the ability to retain cell state for more than ten years \[36\]. These features pave the way for PCM to be mass produced and commercialized \[6, 21\]. Compared to DRAM, PCM has better scalability,
CHAPTER 1. INTRODUCTION

comparable read latency, but lower per-cell endurance. The ITRS 2015 Roadmap [36] highlights endurance and reliability requirements of non-volatile memories as a major barrier to overcome, in order to leverage this technology for high density memory.

Each PCM cell can endure only a limited number of writes [6, 47, 50, 53, 65]. The exact cell endurance depends on the PCM type (i.e., Single-Level Cell (SLC), Multi-Level Cell (MLC), and Triple-Level Cell (TLC)), the fabrication technology, and the operating temperature. When a PCM cell reaches its endurance limit, it loses its ability to store new data (usually resulting in a stuck-at either ‘0’ or ‘1’). As a result, cell wear-out significantly degrades system reliability and performance by gradually degrading memory capacity [50, 67]. Unfortunately, error recovery techniques presently employed in DRAM memory are not a great match for PCM, since unlike DRAM, the errors are permanent, and the number of failed bits accumulates over time.

ITRS [36] projects that cell endurance will continue to be a challenge in future PCM generations for two reasons. First, future PCM chips will likely store multiple bits per cell in an effort to increase bit density when targeting PCM as DRAM replacement. Multiple resistance ranges in MLC PCMs are used to realize multiple states. Write operations in MLCs are performed using iterative programming and are verified using subsequent read verification [6, 43], which ensures that the resistance of each cell is within a safe region. The verification pulses are automatically injected into a closed-loop circuit within a PCM chip, which further exacerbates the wear-out problem [6]. In current prototypes, the reported cell endurance for SLC PCM is around $10^7$–$10^9$ writes, while it is $10^5$–$10^6$ in MLC PCM [6]. Second, as memory cell sizes shrink with technology improvements, we expect lower cell endurance because the junction between the heating electrode and GST becomes weaker, and current density will increase [69].

Therefore, the first part of this thesis will focus on the limited write endurance issues associated with PCM. To this end, the first finding of our work is to introduce a reliability/endurance mechanism for improving PCM reliability. The proposed scheme decouples error location information from error correction, therefore, all the metadata bits can be used for error correction and as replacements for faulty bits to tolerate many faults in a memory block. Detecting a fault and identifying its location in a block are performed by imposing one extra write on each access to faulty blocks.

A second issue addressed in this thesis is the fact that the metadata for error recovery is not utilized efficiently with current error recovery schemes for PCM. Motivated by this observation, we propose a simple, yet very effective, technique called block cooperation to extend memory lifetime. Block cooperation technique can be incorporated on top of different error correction schemes to
realize a durable and dependable memory system.

Despite the attractive features of PCM, aggressive scaling of PCM reduces the cell-to-cell distance, which results in non-negligible thermal interference between cells. Experimental results show that thermal disturbance is one of the major reliability concerns in PCMs, especially as we start to reduce the size of these devices [63]. When writing to a cell, adjacent cells are thermally disturbed, such that an unintentional state may be induced. This inadvertent bit flip during the write process is referred to as a Write Disturbance. This phenomenon has been reported in by Lee et al. [52] in 54 nm technology, and has become a dominant issue below 20nm [20]. For reliable operation, write disturbance issues in PCM need to be reduced or eliminated. Another contribution of this study is to address the negative impact of write disturbance on the system metrics. We proposed two cost-effective write disturbance mitigation techniques, while incurring minimal impact on the lifetime and energy consumption of PCM.

The third major focus of this thesis addresses the relationship between security and lifetime in NVMs, where security is a key aspect of a dependable memory system. While nonvolatility is a desirable feature to save energy, it creates security vulnerabilities since data values persist after system power-off. Data stored in NVMs can be secured using data encryption. However, side effects imposed when performing memory encryption include excessive bit writes, which will drastically reduce NVM lifetimes and increase energy consumption. Considering current solutions for secure NVMs, still there is a considerable gap between encrypted vs. unencrypted NVMs, in terms of lifetime and energy consumption. Our proposed architecture exploits standard counter-mode encryption to maintain security, while keep track of different versions of the modified data in each memory writeback. Selective re-encryption is performed based on the history of the modified data in cache lines, limiting the number of bit writes. Our proposed scheme for secure NVMs improves memory lifetime by 53% (2.87X) as compared to a state-of-the-art (full-line encryption) scheme, incurring only 6% more energy versus an unencrypted memory system.

### 1.0.1 Summary of Contributions

In this thesis, we focus on various aspects of dependability of NVMs, particularly reliability, durability, and security. To this end, we explore and propose several mechanisms at the architecture level, and evaluate the implications of our schemes on a number of important metrics of the system, including performance, energy consumption, and wear-out.

The key contributions of this thesis are summarized below:
CHAPTER 1. INTRODUCTION

- **A Reliability/Endurance Mechanism for Advancing PCM (REMAP).**
  PCM can only endure a limited number of writes – wear-out is a very important class of fault model in PCM. Error-correcting-codes (ECCs) are a must to provide reliable operation, but current ECCs are only sufficient to recover from a small number of errors. We propose REMAP to address this issue. Remap eliminates the overhead of storing error locations in the metadata, and as a result, can correct many more errors. However, the proposed REMAP scheme imposes an extra write to find the error locations whenever a faulty block is accessed. To alleviate the adverse effects of the extra write on PCM lifetimes, we enhance our proposed REMAP scheme by incorporating three different mechanisms: i) static and dynamic partitioning schemes, ii) fault location caching, and iii) employing REMAP as second layer of defense. Our enhancements significantly improve REMAP and increase PCM durability.

- **Implementing a memory lifetime analysis framework for NVMs lifetime.**
  To evaluate the proposed REMAP, and compare it against several state-of-the-art error recovery schemes for NVMs, we implemented a framework to provide memory lifetime analysis. Our framework provides memory lifetime analysis through Monte Carlo and trace-driven simulations. Simulation results show that the proposed REMAP is capable of boosting PCM lifetime by 56% on average (up to 78%) as compared to our baseline. Furthermore, we evaluate the performance loss of REMAP, assuming different error rates, and show this lifetime achievement comes at a negligible performance cost.

- **A block cooperation mechanism to increase metadata utilization, and enhance NVM durability.**
  Using our memory lifetime analysis framework, we found that the metadata is under-utilized for error correction due to the fact that memory addresses are not accessed uniformly. To reduce the page waste caused by early block failures, we propose a block cooperation mechanism, which is modeled using statecharts in this thesis, and allows us to evaluate our scheme when implemented on top of other error correcting mechanisms. We show that by adopting our block cooperation at a single level (or multiple levels) on top of ECP and Aegis, we can increase memory lifetimes by 28% (37%), and 8% (14%) on average, respectively.

- **Two cost-effective solutions to mitigate write disturbance in dense PCM.**
  Dense PCM suffers from specific fault model named write disturbance. In this thesis we evaluate state-of-the-art schemes used to mitigate write disturbance in dense PCM. Based
CHAPTER 1. INTRODUCTION

on this analysis, we propose two cost-effective solutions to reduce the probability of write disturbance. Our solutions come with few side-effects on other memory system metrics. The first technique is based on data encoding, and tries to reduce the number of vulnerable data patterns when writing data to main memory. The second technique detects vulnerable cells, and overwrites them if their occurrence is below a set threshold. The proposed techniques are general and can reduce the average number of writes by 49% over traditional schemes, while incurring minimal impact on PCM lifetime

- **Proposing Nacre, a write-efficient security mechanism to enhance NVMs durability.**

  A security mechanism that is particularly tailored for non-volatile memories is required to mitigate the negative effects of encryption on NVMs. For this purpose, we propose Nacre, that attempts to bridge the gap between encrypted and unencrypted NVMs. Nacre limits the number of bit writes through selective re-encryption of data segments, based on tracking the history of data modifications in cache lines. Our preliminary results show that Nacre is capable of reducing the bit flips in secure NVMs significantly. More analysis on new benchmarks, comparing with previous schemes, and evaluation on other metrics of system such as energy consumption and memory lifetime are the parts of ongoing work that we will address in this thesis.

1.0.2 Organization of the thesis

The main focus of this thesis is to design algorithms at an architectural level to realize dependable NVMs as main memory. The thesis proposal is organized as follows. Chapter 2 presents the background information on PCM basics and our fault model. Prior work related to this thesis is presented in this chapter. In Chapter 3, Chapter 4, Chapter 5 and Chapter 6 we introduce our proposals to realize dependable NVMs as a main memory. Chapter 3 describes REMAP, a reliability/endurance mechanism for advancing PCM lifetime. By decoupling error correction from error detection, REMAP is capable of tolerating more errors as compared to the previous error correction schemes. In Chapter 4 we introduce a block cooperation scheme to utilize metadata effectively, and further enhance memory lifetime. Chapter 5 tackles write disturbance in very dense and small geometry PCM devices. In this chapter we present two cost-effective techniques to mitigate write disturbance problem in PCM. Chapter 6 describes Nacre, a write-efficient security mechanism designed for NVMs. Finally, Chapter 7 discuss our ongoing work and the path to complete this thesis.
Chapter 2

Background

We begin by describing the physics of a PCM device, and how read/write operations are performed. We also describe fault models and wear-out characteristics of PCM, including resistance drift and write disturbance. We continue by describing fault detection mechanisms and the basics of a PCM-based Dual In-line Memory Module (DIMM). We review how a comparison write is commonly used as an effective technique to reduce the write side-effects in PCM. Finally, we review prior work targeting reliable and secure PCM-based main memory.

2.1 PCM Basics

PCM stores data in a chalcogenide alloy of Ge$_2$Sb$_2$Te$_5$, called GST, in either a high-resistance amorphous (RESET) state or a low-resistance crystalline (SET) state. Figure 2.1 (a) shows the structure of a PCM cell, which is composed of top and bottom electrodes, a heater, and a programmable area (i.e., the phase change material). The phase of the material is changed between the amorphous and crystalline states using the heater. Figure 2.1 (b) shows the operation of a read and a write in PCM. To RESET a PCM cell, a rather short, high voltage, pulse is used to heat the PCM above its melting point ($T_{melt}$). To SET a PCM cell, a continuous, but low voltage, pulse is used to keep the PCM at slightly lower temperature ($T_{cryst}$), transforming the PCM material into it’s crystalline state. The read operation is performed by applying a low voltage pulse to the material. Then, a SET or RESET state is determined by sensing the resistivity of the cell.
CHAPTER 2. BACKGROUND

2.2 PCM Fault Model

Generally, the PCM fault model can be categorized as hard and soft faults, which we describe next.

2.2.0.1 Hard Faults

PCM cells stop switching between binary states after a number of writes. This phenomenon is specifically due to heat stimuli and a melt-quenching process, resulting in switching between binary resistive states. As more and more state transitions occur, at some point the PCM loses its ability to switch, leaving the bit stuck at either a low-resistance state (SET) or a high-resistance state (RESET), referred to as a SET-stuck failure (SSF) or RESET-stuck failure (RSF), respectively. SSFs and RSFs occur for different reasons. An SSF occurs due to germanium (Ge) depletion inside the programmable area, which affects the quality of GST in terms of its resistance level [47]. RSF, on the other hand, occurs because of the heated electrode detaching from the GST, resulting in a stuck-at fault [47]. An SSF and an RSF in a PCM cell are shown in Figure 2.2 (b) and Figure 2.2 (c), respectively. While the former can be recovered to some extent by utilizing a reverse electric field [53], the latter is unrecoverable.

In this thesis proposal, whenever we refer to hard or wear-out faults in PCM cells, we mean either SSF (stuck-at ‘1’) or RSF (stuck-at ‘0’). Note that in both cases of stuck-at faults, the cell cannot be reprogrammed, but can be read using normal read circuitry in PCMs. In contrast to DRAM, where bit errors are random, temporary, and very scarce, PCM errors are permanent and the number of errors in PCMs increases over time. The mechanisms used for checking and redundancy
CHAPTER 2. BACKGROUND

Figure 2.2: PCM cell when it is (a) SET-stuck failure (SSF), and (b) RESET-stuck failure (RSF).

in current DDR3 and DDR4 DRAMs (such as SECDED) are insufficient for PCMs, mainly due to having different error profiles. The errors that occur in DRAM are primarily transient, and can occur at any time, whereas stuck-at errors in PCM are permanent and only occur after write operations.

2.2.0.2 Soft Faults

Similar to DRAM technology, PCMs suffer from soft faults. However, the sources of soft faults in PCMs are different as compared to DRAM. Soft faults should be mitigated wisely in order to design a reliable and robust memory system. Two types of soft faults are present in PCMs:

i) Resistance drift faults. Resistance drift is not a problem for SLC PCMs. However, in MLC PCMs, cell resistance increases over time. This can lead to intermediate states crossing the resistance boundary in MLC PCMs, resulting in unreliable read-out. Different techniques have been proposed to tolerate resistance drift. We can exploit reference cells and data encoding [62], as well as scrubbing mechanisms [8], to address drift faults in MLC PCMs.

ii) Write disturbance faults. When writing to a cell by applying voltage pulses, adjacent cells may thermally disturbed. These neighboring writes can lead to a neighboring cell’s state to change. This inadvertent resistance change in a PCM cell is referred to as a write disturbance. It can occur when an idle cell is in the amorphous state (i.e., zero binary), and a neighboring cell is programmed to be RESET. The idle cell’s state can partially altered to the SET state, if it experiences a temperature higher than that of the crystallization point ($T_{\text{cryst}}$), as shown in Figure 2.1 (b). Therefore, a write disturbance occurring in a memory block is highly dependent on the data pattern – both the previous data stored in the block, and the values in the new data that is being written. This problem is more pronounced in small device geometries and dense PCMs, where the inter-cell distances can be very small [27, 41, 52]. Trading off capacity for reliability by introducing
CHAPTER 2. BACKGROUND

Figure 2.3: The baseline architecture of a PCM-based DIMM.

thermal bands along cell lines can effectively reduce write disturbance. Data encoding \[27, 82\] and data compression \[41\] have also been proposed as solutions to reduce the probability of a write disturbance.

2.3 PCM-based Memory Module

Figure 2.3 shows the baseline architecture of a PCM-based Dual In-Line Memory Module (DIMM) that we consider throughout this thesis. A PCM rank consists of nine 8-bit chips, where the ninth chip stores ECC. A rank is composed of multiple banks, in order to perform multiple memory accesses independently. One logical bank is interleaved across all the chips in the DIMM and each sub-bank contributes a part of the requested memory block. Assuming a 64B last level cache line size, it takes 8 burst accesses to read or write-back data from/to main memory. The standard DIMM has a 64-bit data path, along with an extra 8-bits for metadata. On memory requests, the ECC information is also sent to the on-chip memory controller to provide error detection/correction.

Write operations consume a considerable amount of energy in PCMs. The number of PCM cells that can be written, in parallel, is limited (e.g., 128 cell writes). Therefore, PCMs employ a differential write or Data Comparison Write (DCW) \[95\] to limit the number of required cell writes. DCW first reads the old data, compares it with new data, and only writes to the modified bits in the cache line. By adopting DCW in PCMs, we can reduce write energy and cell wear-out significantly, prolonging PCM lifetimes, as well as improve performance.

Since the memory controllers on the processor side have little knowledge of the status of each device, a new chip is proposed to sit between the controller and underlying memory subsystem to decouple them. This design avoids extra bus contention, improving scalability \[85\], performance \[5\], and power efficiency \[3\]. This chip, named a bridge \[29\], receives device-independent commands.
CHAPTER 2. BACKGROUND

from the memory controller and generates optimized device-specific commands.

We have a number of design choices to make when selecting how to implement error correction logic for PCM. ECC can be stored in the PCM chips, as proposed in SAFER [73]—an error correction scheme proposed for PCMs. Despite providing low-latency error recovery, replicating ECC logic for each chip is not very economical. The other choice is to place ECC logic within the bridge. This design makes error correction transparent from the perspective of the memory controller and allows near-to-DIMM data recovery without involving the memory bus. The ECC-on-SIMM [23], and advanced-memory buffer (AMB) [79], which were both proposed by IBM, are similar approaches for providing self-contained ECC logic on the memory module. The final design choice, which has been mainly adopted by industry, is to place the ECC logic within the memory controller.

As shown in Figure 2.3, within PCM chips, there is an embedded read-modify-write (RMW) circuit which performs a bit-wise comparison of the stored and updated data [95]. The RMW circuit ensures that we modify only the specific bits that we want to change, achieving longer lifetimes and consuming less energy.

The read verification [6,43] is also a part of the PCM technology standard, embedded into the logic of PCM chips. The extra read ensures that the resistance of each cell is within a safe region and is used for error detection as well. The verification pulses are automatically injected after each write operation. If the memory block is faulty, the bridge can issue extra writes to locate the faulty cells.

2.4 Related Work

In an attempt to relax the problem of limited lifetime in PCM, prior studies have proposed techniques that exploit write buffers [50,67], use compression [10,26,39,94], and use encoding schemes [9,38] in order to reduce write traffic in PCM-based memory systems. In parallel, the controller may use wear-leveling techniques (such as Start-and-Gap [65]) in order to uniformly spread the write traffic over all memory blocks, which delays wear-out of some of the memory blocks if they are under high write pressure. With all these fault avoidance techniques, when a cell becomes faulty, we need a mechanism to perform error correction to extend memory lifetime. Next, we review the prior work on error recovery scheme for NVMs. Then, review the prior work on reducing the write disturbance in PCM. Finally, we discuss related work targeting secure NVMs.
CHAPTER 2. BACKGROUND

2.4.1 Prior Work: Hard-Fault Tolerance in PCM

Hard errors are permanent and increase over time. So the mechanisms found in current DRAMs (such as Single Error Correction, Double Error Detection–SECDED) are insufficient for PCMs due to their very different error profiles. Prior work proposed error correction schemes specifically designed for PCM to improve memory lifetime. We classify these schemes into the following three categories.

**Category 1 – Replacing faulty bits with healthy bits.** The basic idea is to find the location of bit failures and to replace them with healthy bits. The Error Correction Pointer (ECP) scheme [72] is the pioneering work in this category, where for every faulty bit, the scheme keeps one pointer and one replacement bit (forming one ECP entry). Error correction is performed by restoring the correct value of the faulty bit (the location where the ECP points) with the replacement bit. Assuming a standard DIMM with one ECC chip for every eight data chips [37], ECP can correct six faulty bits, irrespective of the location of the faulty memory cells in a 64-byte data block. This is known as ECP6, and we will refer to it as such throughout this thesis.

Qureshi [66] proposes the Pay-As-You-Go (PAYG) error recovery scheme in order to significantly reduce the storage overhead of ECP. Leveraging PAYG, for each data block, one dedicated ECP entry is allocated. To correct more bit failures, PAYG maintains additional ECP entries in a global structure, enabling dynamic management and reducing the storage overhead. PAYG enjoys a $3 \times$ reduction in ECP storage overhead, with only minimal performance degradation.

REMAP [83] uses all the metadata space for replacing faulty bits. The locations of failed memory cells are identified by extra write operations, instead of storing a pointer to each failed cell. Therefore, the number of correctable errors increases. To alleviate the overhead of extra writes on memory lifetime, REMAP uses static and dynamic partitioning.

**Category 2 – Partitioning data blocks and Bit-wise Inversion.** The schemes in this category mask errors by storing data in their inverted forms (e.g., storing a ‘0’ instead of a ‘1’ into an stuck-at RESET cell). A necessary condition to be satisfied in order to use inversion for masking errors is that each faulty cell needs to be in a different partition. SAFER [73] partitions the data block dynamically to ensure that each partition has at most one faulty bit, and decides whether to store data in its original or complement form. Aegis [28] exploits the same approach, but uses a more efficient data block partitioning scheme – Aegis corrects more faulty bits with fewer partitions, as compared to SAFER. RDIS [57,59] works in the same way, but is more efficient than SAFER when the number of faulty bits grows beyond six. However, there are some error patterns that RDIS may fail to recover...
CHAPTER 2. BACKGROUND

from when the number of errors is even greater than three.

Compared to the schemes in the first category, all three implementations in the second category can tolerate more errors by removing the need for pointers in metadata. However, these schemes usually suffer from two issues. First, due to the lack of tracking the location, the solutions in the second category usually impose extra write overhead, which negatively impacts memory performance and endurance. The second issue relates to increased design complexity when implementing complicated partitioning algorithms.

Category 3 – Pairing the Faulty Blocks. When a memory block (either a data block or page) becomes faulty, instead of discarding it, we can pair it with another failed block to restore data correctness using redundancy. Dynamically Replicated Memory (DRM) \[35\] implements this concept by pairing two failed memory pages whose faulty bits are not at the same offset. Using this scheme, DRM delivers better memory system lifetime by gradually degrading the available storage. Free-p \[91\] is another scheme where each memory block that has many faulty bits, stores a pointer to a replacement block, reducing the need for a pointer field as required in the first category. The pointer is replicated multiple times to tolerate errors. Zombie \[9\] extends the memory lifetime by reusing the healthy blocks of a discarded page that is visible to software. Zombie significantly improves endurance, but requires complicated bookkeeping at a hardware level.

Chen et al. \[16\] proposed dynamic redundancy using mirroring and parity pages. In the mirroring scheme one redundant (mirror) bit is selected for every data bit by grouping two pages. The mirror page is selected in a way that there is no fault in the same bit location of the corresponding faulty page. In the parity page scheme, N faulty PCM pages form a group together to tolerate errors. This scheme imposes off-chip DRAM buffer and on-chip cache structure overheads for page mapping.

2.4.2 Prior Work: Write Disturbance

For reliable operation, write disturbance issues in PCMs need to be reduced or eliminated. At a layout level, allocating sufficient inter-cell thermal bands can effectively reduce write disturbance. For instance, in a 20 nm PCM prototype \[20\], they proposed leaving 20 nm of extra space to provide a thermal band along word-lines to prevent a write disturbance. This approach eliminates write disturbance, at the cost of reducing chip capacity. Exploiting a built-in verification circuit that issues verification reads \[19\] after each write ensures that memories provide reliable write operations in PCMs, without any modifications for disturbance mitigation. The main drawback of this class of
CHAPTER 2. BACKGROUND

Schemes is a dramatic performance impact, especially when the probability of a disturbance is high. Detecting potentially disturbed cells, and overwriting their data values, is another approach to eliminate write disturbance [41]. This scheme relies on a write comparison [95] algorithm that compares the new data block with the old one, identifying potentially disturbed cells. This approach has some negative side-effects. First, writing to disturbed cells by performing a RESET dissipates 25% to 150% more power (as compared to a SET pulse). Second, the limits on current delivery of the charge pumps in the PCM peripheral determines the number of parallel cell writes. For example, in a 20 nm PCM prototype, 128 cells can be written in parallel [20]. As we increase the number of written cells, we increase the number of required write slots to update the entire data block (i.e., 512 bits). Third, a small modification in a data block can result in a domino effect, generating multiple unnecessary cell writes to the data block, which increases cell wear-out.

Jiang et al. proposed a Data encoding-based INsulation technique (DIN) [41] to reduce the number of disturbed cells. DIN mitigates write disturbances by reducing the number of vulnerable data patterns. This scheme first compresses data blocks, and then exploits a simple encoding to expand the data with a disturbance-friendly data pattern. Finally, a Bose-Chaudhuri-Hocquenghem (BCH) code is used to correct up to two errors caused by a write disturbance. Despite imposing considerable complexity, DIN is not a general solution and its effectiveness is highly dependent on the compressibility of the workload. Moreover, by changing the pattern of data, it can reduce the effectiveness of comparison writes, and increase energy consumption.

The schemes discussed above mitigate write disturbance along the memory word-lines. Writes may disturb adjacent lines in a PCM if the distance between bit-lines is very small. SD-PCM [88] addresses write disturbance along bit-lines by proposing LazyCorrection. This technique uses DIN to reduce disturbance along word-lines, and exploits unused Error Correction Pointer (ECP) [72] entries to correct disturbed cells in adjacent lines, whenever possible. LazyCorrection requires a low density ECP chip to avoid write disturbance problems in the correction chip. Moreover, to further reduce performance loss, two pre-reads to adjacent lines associated with the written line are scheduled in idle periods. These pre-reads fetch the contents of the adjacent lines into a buffer, which triples the size of the write buffer.

2.4.3 Prior Work: Secure Non-volatile Memory

In state-of-the-art memory counter-mode encryption [7, 31, 45, 76] a counter is assigned to each memory line. The counter is incremented on each write to guarantee that a unique pad is
generated on each encryption. One bit modification can cause many bit flips over the entire line. Therefore Block Level Encryption (BLE) \cite{48} and DEUCE \cite{92} have been proposed for NVMs in order to reduce the number of bits modified. BLE \cite{48} uses multiple standard AES units (16 bytes) and assigns a separate counter per AES unit to reduce the write overhead. BLE applies re-encryption only on the modified block, equal in size to the AES unit. DEUCE \cite{92} operates at a finer granularity (2 bytes), as compared to BLE, in order to effectively reduce the extra bit writes.

SECRET \cite{80} integrates energy masking with memory encryption to reduce the energy consumption of multi-level cell (MLC) NVMs. SECRET partitions cache lines into eight 8-byte words, and avoids unnecessary bit writes by detecting unmodified zero-words (which are frequent). i-NVMM \cite{17} proposes selective encryption according to the data usage: hot data is kept unencrypted, and cold data is detected periodically and encrypted in the memory. The main shortcoming of i-NVMM, besides not encrypting all of memory, is its vulnerability against bus snooping attacks.

Silent Shredder \cite{7} showed that a large number of the writes that occur in main memory are due to the operating system zeroing out physical pages before remapping them to new processes. By modifying the initialization vector used in standard counter mode encryption, Silent Shredder completely removes the extra writes due to zeroing out pages. CryptoComp \cite{40} exploits memory compression to confine the bit flips within a compressed cache line. However, adding the overhead of compression on top of encryption time may not be tolerable in many applications. SEDURA \cite{56} combines encryption with wear-leveling algorithms in PCMs to reduce the side effects of encryption on lifetime.

Memory compression mechanisms can also be used to save energy and/or increase memory lifetimes in encrypted \cite{40}, or unencrypted \cite{39,54,61} NVMs. Space Oblivious Compression \cite{54} uses in-place and lightweight compression that is tailored to reduce a NVM’s energy, rather than saving space. Jadidi et al. \cite{39} exploit memory compression, along with an intra-line wear leveling mechanism, to increase the effectiveness of error correcting codes in NVMs, increasing memory lifetimes. CompEx++ \cite{61} integrates compression with expansion coding to simultaneously improve energy, latency, and lifetimes of MLC NVMs. While this work exploits memory compression to reduce bit flips in unencrypted memory, CryptoComp \cite{40} exploits memory compression to confine the bit flips to a compressed cache line, minimizing the number of AES standard blocks used to encrypt the memory accordingly. However, adding the overhead of a compression on top of the encryption time may not be tolerable in many applications, considering that decompression latency cannot be hidden.

The techniques used in Silent Shredder \cite{7}, SECRET \cite{80}, SEDURA \cite{56}, and Crypto-
CHAPTER 2. BACKGROUND

Comp \cite{40}, are orthogonal to the architecture support for data versioning that we employ in our proposed scheme in Chapter 6 and can be used together to improve PCM lifetimes.
Chapter 3

REMAP: A Reliability/Endurance Mechanism for Advancing PCM

To tackle the endurance problem for future memory systems, we propose REMAP, a Reliability/Endurance Mechanism for Advancing PCM. REMAP is designed to efficiently use all the metadata bits as replacements for faulty bits. Exploring PCM reliability solutions is an important in terms of the future of PCM technology, increasing the changes that industry will adopt this technology in the future. The key idea of REMAP is to decouple error location information from error correction metadata, therefore, all the ECC bits can be used for error correction and as replacements for faulty bits to tolerate many faults in a memory block. Fault detection and location in a block are performed by imposing one extra write on each access to faulty blocks.

Despite tolerating many hard-errors, extra writes in REMAP could cause significant degradation in terms of memory performance and memory lifetime. Hence, it is important to develop mechanisms to address the potential downsides of REMAP. To this end, we propose three complementary techniques. Our first approach leverages partitioning, with the goal of lowering the number of bits that are experiencing extra writes. Our partitioning scheme helps the memory controller identify which partitions contain errors, localizing the extra writes to only erroneous partitions. The memory controller can perform static or dynamic partitioning. The former approach (i.e., static) performs error recovery with lower design complexity and lower overhead, while the latter approach (i.e., dynamic) is adaptive to the fault patterns, but comes with more complexity and higher overhead. The second technique, fault location caching, stores fault location information of recently-accessed memory blocks in a small cache to completely avoid the extra writes (if they are
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

in the cache). Equipped with caching, REMAP can significantly reduce the overhead of the extra reads and writes, at the cost of adding a small buffer. The third approach pursues a complementary alternative error correction scheme to remove or lessen the impact of the extra writes when the number of errors is small, and only exploits REMAP as a second layer of defense when a block experiences a large number of errors. We evaluate the PCM lifetime benefits of REMAP as compared to three state-of-the-art error correction schemes. Our analysis evaluates REMAP using both real workloads and Monte Carlo simulations. We find that REMAP can improve PCM lifetime by 56% on average (up to 78%), as compared to the previously-proposed state-of-the-art scheme.

The remainder of the chapter is organized as follows. Section 3.1 describes our REMAP scheme. Section 3.2 provides three improvements to our vanilla REMAP scheme, attempting to mitigate the extra write overhead associated with REMAP. Section 3.3 describes our trace-driven lifetime evaluation framework, as well as workloads selected for evaluation. Section 3.4 reflects on the hardware and delay tradeoffs associated with these schemes. Finally, Section 3.5 concludes the chapter.

3.1 REMAP: A Reliability/Endurance Mechanism for Advancing PCM

Since stuck-at errors only occur during writes, to detect worn-out bits, a read verification is performed after each write [19]. Read verification checks if a programmed cell is in the desired state (resistivity), and this extra read is necessary due to the uncertainty of the heating process. Therefore, for write operations, the first data block is stored in temporary storage, and is later programmed into selected cells. After programming, bit-by-bit comparisons are performed to make sure the original and stored values are the same. If there is no difference, the verify read operation indicates a program success. Otherwise, it indicates a program failure.

Read verification logic is already implemented in MLC memories (required by the write algorithm [6]), and used in previous studies [9, 28, 59, 72, 73] for error detection. Similar to our work, read verification is performed, and whenever an error is detected, an error recovery mechanism is initiated to correct the error.

Bit replacement schemes are capable of tolerating a limited number of errors, because many of the metadata bits are devoted tracking error locations. For example, to support ECP6 [72], nine out of ten bits are required to point to the location of error, and only a single bit keeps the actual value of the faulty cell. Alternatively, partitioning and inversion schemes are able to tolerate many more errors by exploiting complex dynamic partitioning on a block basis, determining the
best inversion scheme for the data for each partition in order to mask errors. In this case, the exact location of the errors is not clear, but the location of the partitions that contain errors are encoded in the metadata. The key idea behind REMAP is to eliminate the overhead of storing error locations in the metadata, which can allow us to correct many more errors.

In order to avoid the storage overhead for maintaining the fault location, an alternative mechanism is required to find the location of faulty cells. we can SET all the PCM cells to find stuck-at 0 bits, or similarly, RESET all the cells to find stuck-at 1 bits. A more effective approach is when we detect a block as faulty, by toggling all the bits on the write operation and subsequent read verification, locate the faulty cells. Using this approach, finding the fault locations in faulty blocks only incurs an extra write for the read and write operations.

By eliminating the need to record the fault location, we can devote all the metadata bits to error recovery of faulty cells. However, this proposed change has a three complications. First, imposing an extra write is not an attractive solution, since PCM will wear-out faster with these extra writes. Second, despite eliminating the storage overhead for locating the fault, the cost reappears as a delay overhead of the extra write for locating faults in faulty blocks. Third, is it a good choice to devote all the metadata budget to error correction without reserving any bits to locate the faulty bits in a data block? In the following sections, we address these concerns when using REMAP.
3.1.1 Example: Error Correction with REMAP in Read/Write Operation

Main memory read and write operations are generally performed at a page level granularity. Due to the limited endurance of PCMs, writes are performed at the granularity of the processors’s last-level cache line or block (e.g., 64B) to minimize the number of writes to pages [67].

A write operation performed in a faulty memory block is shown in Figure 3.1. 1 represents a 512-bit data block, which is being written into a particular physical memory block in the PCM-based DIMM (refer to Figure 2.3). However, the verification read detects that one or multiple bits are in error. A verification read is performed after each write to make sure a data block is written correctly in the PCM cells. As shown in 2, four faulty cells are in the memory block, with one of them masked (bit 507), but the others are visible and detected via read verification.

REMAP requires us to identify the location of the faulty bits, therefore, we first write all the bits in their complement form, followed by a read verification. The extra write helps us to identify all faulty cells in the memory block. As shown in 3, bits 0, 507, 509 and 510 in the given memory block are either stuck-at SET or RESET. The contents of these faulty bits are stored in metadata as a Fault Bit Stream (FBS), in order as 4 shows. The first bit in the metadata (the Faulty Block Indicator or FBI) is used to indicate whether the data block is faulty (or not), the second bit (Inv) indicates whether the data is stored inverted (or not), and the remaining 62 bits are used for error recovery. Finally, data block 1 (as well as the generated metadata 4) are written to the memory block. Note that, when using the read-modify-write circuitry in the PCM chips (refer to Figure 2.3) [95], we can remove the first write in step 3 when the memory block is faulty. In other words, two writes are imposed if a block experiences a first fault, otherwise if it has already experienced a fault, only one extra write is imposed for locating the error.

Assume that a read access to the same memory block occurs after the write access. The metadata should contain all the required information to reconstruct the correct data. Figure 3.2 shows the steps of the read operation. In 1, the data and metadata are read. We detect that the memory block contains faulty data by checking the FBI bit. To find the error locations, we write the complement of all the bits and verify the values with a read 2. The locations of the faulty cells are revealed in this step, since we will identify bits that did not toggle. When an error is found, the bridge chip in the DIMM needs to correct the data block by replacing the correct values using the metadata 3. Note, that by writing the complement of all bits, the current data is inverted, and then written, using the Inv bit to denote this format change so we can know to to invert the data on the next read.

The faulty cells in this example are bits 0, 507, 509, and 510. They are replaced by a FBS
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

![Diagram of data block and metadata](image)

Figure 3.2: Read operation from a faulty memory block.

in bits 58–61. Note that the locations as well as the number of faulty cells are obtained by issuing an extra write operation, and that only the contents of the cells are stored in the metadata section. Decoupling the location of errors from the metadata potentially results in a recovery of up to 62 errors. However, the extra write overhead required to find the location of the errors may negate any of the benefits. The impact of the write overhead is negligible when the number of faulty blocks is small, but as the number of faulty blocks increases, system performance also degrades gradually. SECDED, ECP [72] and PAYG [66] schemes save the locations of errors and avoid the extra write, but can only tolerate a limited number of faults. SAFER [73], RDIS [59] and Aegis [28] are similar to REMAP in that they cannot avoid the extra write overhead to reveal partial fault locations. However, accepting this overhead, we can tolerate a larger number of faults than the previous schemes, and potentially provide better reliability. In the following, we conduct a set of Monte Carlo simulations to understand how REMAP performs as compared to the other competitive fault-tolerance schemes.

3.1.2 Experimental Setup

In the following, we describe our simulation platform, simulated error correction schemes, and analyze our simulation results.

Simulation Platform. For each scheme, we model a 4K 4KB-pages, where each 4KB page contains 64 memory blocks. Each memory block uses 512 memory cells for data and 64
memory cells for metadata, similar to the standard PCM-based DIMM shown in Figure 2.3.

To model the lifetime of our memory system, we have followed the methodology adopted in previous studies \cite{28,59,72,73,84}. A value is assigned to each PCM cell that represents the number of writes before the cell wears out. The lifetime is assumed to follow a normal distribution, with a mean ($\mu$) of $10^8$ and a range of Coefficients of Variation (CV) values of 0.2, 0.25, and 0.3. The CV parameter models the impact of process variation in the PCM cells. The major sources of process variation in PCM include: i) heater thickness, ii) bottom electrode contact diameter, and iii) GST layer thickness \cite{94}. When reducing the size of the cells, PCM experiences a higher degree of process variation, which negatively affects endurance.

Similar to previous work \cite{28,59,72,73,84}, we assume perfect wear leveling across the memory blocks, such that the wear-out occurs uniformly across a block. This assumption is in line with wear leveling techniques previously proposed for PCMs \cite{2,65,74}. The number of read accesses and the number of write accesses are considered equal, and the probability of a value change for each bit is assumed to be 0.5 throughout the simulations. Whenever the error correction scheme cannot correct the faults in a memory block, the whole page containing the memory block dies and is removed from the accessible physical memory address pool by the operating system. By increasing the number of dead pages, the size of the physical memory is reduced, resulting in more wear and stress on the remaining pages. Simulation continues until all pages die, and a given error correction scheme cannot recover from a failure any longer.

**Simulated Error Correction Schemes.** We consider a number of error correction schemes in our evaluation. Note that all of our configurations assume 64-bits of metadata for every 512 bits.
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Figure 3.4: Probability of having a different number of faults in data blocks with CV=0.25 for: (a) REMAP and (b) Aegis 23×23 schemes.

Our configurations for SECDED, ECP6 [72], SAFER32 [73], Aegis 23×23 [28], Aegis 17×31 [28] satisfy this overhead requirement for 512 bits of data. ECP6 tolerates up to 6 faults. SAFER32 uses 32 partitions to be able to tolerate 6 errors deterministically, but can tolerate up to 32 errors probabilistically. Aegis 23×23 and Aegis 17×31 show how 512-bit data is organized in the Cartesian plane. The Aegis scheme tolerates more faults with fewer partitions, as compared to the SAFER32 scheme. We also take into account the wear-out of the metadata bits for each scheme. ECP and SECDED are able to tolerate errors in metadata bits, while SAFER32 and Aegis cannot.

**Result Analysis.** Figure [3.3](#) shows the lifetime of PCM when various schemes are used...
for error correction. Aegis\textsuperscript{17}×31 achieves the longest lifetime for every simulation, independent of CV assumed. The lifetime for Aegis\textsuperscript{17}×31 is reduced by 42\% when the CV varies from 0.20 to 0.30, whereas the lifetime is reduced by 76\% for the SECDED scheme. REMAP shows a 41\% lifetime reduction as the CV varies from 0.20 to 0.30. However, the overall lifetime is not competitive with other schemes. REMAP only performs better versus SECDED, and is comparable to ECP6 when the variance is high. We can tolerate 62 errors using REMAP, potentially increasing the memory lifetime dramatically, but we also face inefficiency in terms of lifetime.

To understand why REMAP does not enjoy longer lifetimes, we report on the number of faults found in each data block after all pages in the system have worn out. The results are presented in Figure 3.4. Figure 3.4a and Figure 3.4b show the distribution of the number of faults found in the worn out block when REMAP and Aegis 23×23 are used, respectively. For REMAP, almost 1.5\% of all the data blocks contain 63 faults, which results in failure of all of main memory. Although we can tolerate many faults using REMAP, almost half of the blocks did not experience any faults (see the first bar in Figure 3.4a).

There is a considerable imbalance in terms of the distribution of the number of faults in the data blocks, which is heavily skewed to the left. For Aegis 23×23, although fewer faults can be tolerated, the distribution of those faults in data block is more uniform. More than 95\% of the data blocks experienced at least one error, and almost 60\% of data blocks contain 3-6 errors. There is a significant difference between how each scheme stresses the memory system. Both schemes impose extra write overhead if a given data block is faulty. However, the number of memory bits that undergo extra writes for Aegis 23×23 is limited, and increases monotonically as the number of faults increases. Moreover, the dynamic behavior of Aegis, which updates partitioning whenever the fault pattern in the data block changes, helps to spread the extra writes more uniformly across all bits. As compared to Aegis, REMAP is less flexible when performing extra writes. All of the 512 bits in the data block are written to find the error location. Writing to all the bits in a data block for locating faults outweighs all the benefits we could obtain from tolerating many number of errors in REMAP. In the following section, we propose novel techniques to alleviate the extra write problem in REMAP.

### 3.2 Extra Write Overhead Alleviation

Next, we describe three different techniques to lessen the impact of extra writes imposed by REMAP on wear-out. First, we present a static and a dynamic partitioning scheme which helps
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

to localize the impact of extra writes on memory cells. Second, we discuss caching fault locations to fully remove extra write overhead on recently accessed blocks. Third, we use complementary alternative error correction schemes to remove or lessen the impact of the extra writes when the number of errors is small, and only exploit REMAP as a second layer of defense when a block experiences a larger number of errors.

3.2.1 Write Overhead Localization Using Partitioning

One of the major drawbacks of REMAP is the extra wear incurred due to the required writes. The write is performed to all cells in the data block, irrespective of the number and the location of faults. With REMAP, all of the metadata bits are devoted to error recovery in order to tolerate many faults. Partitioning the data blocks, in order to inform the bridge chip of which partitions contain errors, helps to localize the extra write overhead. The extra writes are only performed on a subset of the data in the block, not the entire block. Sacrificing a few error correction bits in the metadata, and using those bits to store error information on a partition basis, helps to reduce the overhead of extra writes. Next, two partitioning schemes are presented and evaluated to overcome deficiencies in our REMAP scheme.

3.2.1.1 Static and Contiguous Partitioning

Static partitioning is simple, which helps to dramatically reduce the complexity of the logic added to the bridge chip on the DIMM. Static partitioning is more attractive than using dynamic partitioning schemes, as used in SAFER [73] and Aegis [28]. In static partitioning, the data blocks are logically divided into fixed and contiguous portions. Whenever a fault occurs in a data block, we record its corresponding partition information in the metadata. Therefore, only the cells within the marked partition will experience extra writes for locating faults. As the number of errors increases, more partitions in a data block are affected, which results in increasing write costs. A static scheme tends to be oblivious to the future fault pattern in the data block and cannot adapt itself to take advantage of the fault locations. However, if errors occur close to each other (i.e., they are spatially correlated), the chance of covering more faults in a single partition increases. Alternatively, if the locations of errors are far from each other, the effectiveness of contiguous partitioning is diminished.

There are clearly trade-offs in selecting the right number of partitions, and the resulting number of faults that REMAP can tolerate. As we increase the number of partitions, fewer errors in a data block can be tolerated. However, given the desire to reduce the extra write overhead, we
are pushed to use more partitions. The number of tolerated faults $F_{\text{num}}$, when using partitioning, is shown in Equation (3.1). We account for the size of the metadata $M_{\text{size}}$, and then subtract the number of partitions $P_{\text{num}}$, as well as the extra bits for FBI and Inv. Equation (3.1) shows the number of errors that REMAP can tolerate when static partitioning is used.

\[ F_{\text{num}} = M_{\text{size}} - (P_{\text{num}} + 2) \]  

(3.1)

Increasing the number of partitions reduces the impact of the extra writes. For example, assume that there is one error in the entire data block. If we have 8 and 32 partitions, to locate the faults, we would perform extra writes to 64 and 16 bits of data block, respectively. The average cost of the extra writes $W_{\text{cost}}$ depends on the number of partitions that are faulty $\eta$, which is presented in Equation (3.2). $DB_{\text{size}}$ is the size of the data block (512 bits). REMAP will work well if the faults occur in a small number of partitions.

\[ W_{\text{cost}} = \eta \left( \frac{DB_{\text{size}}}{P_{\text{num}}} \right) \]  

(3.2)

Given that partitioning is static, there is no way to take advantage of the fault pattern to reduce $\eta$. However, $P_{\text{num}}$ is a parameter which can be manipulated to reduce write costs, as Equation (3.2) implies. Nonetheless, increasing $P_{\text{num}}$ reduces the number of tolerated faults, as noted in Equation (3.1). To evaluate the interaction between $F_{\text{num}}$ and $P_{\text{num}}$ in terms of memory lifetime, we explored the design space to find the best number of partitions. We present these results next, comparing the benefits of static partitioning versus dynamic partitioning.

### 3.2.1.2 Dynamic and Non-contiguous Partitioning

Unlike static partitioning, which is oblivious to the location of the faulty bits within a block, dynamic partitioning utilizes the fault locations to reduce $\eta$. In dynamic partitioning, the faulty bits are grouped together to form faulty partitions. The aim of our dynamic partitioning scheme is to minimize the number of faulty partitions dynamically (i.e., $\eta$ factor) in order to reduce $W_{\text{cost}}$ in Equation (3.2). As the number of faulty partitions is minimized, the extra write overhead associated with the REMAP scheme influences a smaller number of non-faulty bits within the data block.

Finding the minimum number of partitions that cover all faulty bits within a data block is, in essence, a classic optimization problem. Using a Cartesian plane to treat the $n$ bits of a data block as a rectangle, the aim is to find the minimum number of lines to cover all the $S$ faulty bits, where $0 \leq S \leq n$. This problem is known as the minimum line covering problem [30]. The maximum
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Figure 3.5: Partitioning with minimum coverage. If (a) slope = 0 is employed, four partitions (0, 1, 2 and 4) are required to cover the four existing faults within the block, but when we select (b) with the slope = 1, the number of faulty partitions is reduced to two (4 and 5).

The number of lines \( l \) (i.e., equivalent partitions) required to cover \( S \) bits is \( \lceil S/2 \rceil \). However, each line can have an arbitrary slope.

While the complexity of the solution to this problem can be bounded by \( O(S \log l) \), it is still costly to implement, since it requires each data block to keep track of not only the number of lines, but also \( k_i \) bits to store the slope of each line \( i \) \( (i = 0, 1, ..., l - 1) \). We can reduce the storage space by limiting all the lines to the same slope (one \( k \) for all \( i \) lines). Here, our problem becomes finding the best slope, \( k \), to minimize the number of lines (partitions) that cover all of the faulty bits, \( S \), within a block.

Figure 3.5 shows a data block with 36 bits, mapped to a Cartesian 6 × 6 plane. This block of data has four faulty bits. As shown in the figure, if the slope of the line is zero, four partitions cover all of the errors. However, as we increase the slope to one, the number of covered partitions is reduced, since only two partitions cover the faulty bits of the block. As the number of faults within the block increases, a new slope has to be calculated to minimize the number of faulty partitions.

While this approach bears some semblance to the approach used by Aegis \([28]\), it tries to solve an entirely different problem. In Aegis, the aim is to find a slope such that each partition only covers a single bit. For REMAP with dynamic partitioning, the goal is to find the slope that places the faults in the minimum number of partitions (minimal coverage).

Similar to the static approach, the number of tolerated faults, \( F_{num} \), varies, depending on
the number of partitions ($P_{num}$) used in the dynamic partitioning. The metadata should maintain one bit per partition to indicate whether the partition is faulty or not, $k$ bits to indicate the single slope associated with all the partitions, and two bits for the FBI and Inv (as described in Section 3.1.1). The remainder of the metadata bits can be used to store the number of faults that can be tolerated. Equation (3.3) is used to compute the number of tolerated faults. The slope $k$ has the range $[0, P_{num})$, in order to guarantee a unique partition for each bit within the block [28]. The slope can be stored with at most $\lceil \log_2 P_{num} \rceil$ bits.

$$F_{num} = M_{size} - (P_{num} + \lceil \log_2 P_{num} \rceil + 2)$$  \hspace{1cm} (3.3)

Dynamic partitioning has to account for scenarios where the best slope $k$ remains the same for a considerable amount of time. In this case, the extra write overhead associated with REMAP can ultimately wear out the non-faulty bits of a partition. We address this concern by constantly changing the slope of the partitions on writes. For example, if two slopes $a$ and $b$ produce the same minimum number of partitions, and if the current slope is $a$, we make sure that the slope $b$ is used for the next write. Changing the slope continuously avoids extra writes, which reduces stress on partitions that were created by using only slope $a$. Additionally, even if $k = a$ provides the best slope for minimal coverage, and $k = b$ is the second best slope for providing minimal coverage, we change $k$ from the previous slope $a$ to $b$. This might expose more bits to extra writes (since more partitions are considered faulty with $b$), but in the long run, this strategy assures us that the extra writes are distributed across different groups of bits, ultimately increasing the lifetime of the data blocks.

Figure 3.6 shows the memory lifetime when REMAP is used with static and dynamic partitioning. The memory lifetime is calculated, varying the number of partitions $P_{num}$ from 10 to 50, and assuming different coefficients of variation. Note that, as Equation (3.1) and Equation (3.3) show, the number of tolerated faults $F_{num}$ differs for the same number of partitions when comparing the static and dynamic partitioning schemes. Even though dynamic partitioning needs to store the slope in metadata, and consequently can tolerate fewer errors as compared to static partitioning, it outperforms static partitioning in all cases when the CV is 0.20. The static partitioning shows better performance as the number of partitions, as well as the variance, increases. To compare results with Aegis 17×31, which obtained the best results across the other schemes, REMAP with static and dynamic partitioning extends the memory lifetime by 786 and 402 millions more writes per page, respectively, when the variance is high (CV=0.3).
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

3.2.2 Fault Location Caching

Caching the location of faults in a small cache helps to avoid extra write overhead, significantly. The location of failed cells in recently accessed faulty blocks is maintained in the cache. SAFER [73] also exploits multi-banked direct-mapped caches to reduce the number of extra writes using a "fail cache". Note that only the visible errors need to be stored in the fail cache, the masked errors are ignored, and we do not lose any error correction capability. Therefore, on every write to the faulty block, given that the pattern of visible errors may change, the location of errors should be updated in the fail cache. If the data block is faulty and its information is available in the fail cache, we can extract all the fault locations from cache to recover the faulty block.

Augmenting REMAP with a fail cache increases the memory lifetime significantly. Figure 3.7 shows the lifetime of the memory using various CV values with REMAP. The lifetime is collected, modeling cache hit rates from 0% (i.e., no cache) to 100% (i.e., perfect cache), in steps of 10%. The intermediate points are generated using interpolation. The lifetime of other schemes are also shown in Figure 3.7. The ECP6, SAFER32, Aegis 23×23, and Aegis 17×31 schemes are evaluated. We also consider using SAFER32 assuming a perfect cache with a 100% hit rate in our evaluations. The X-axis value shown for rival schemes in Figure 3.7 indicates at which cache hit rate REMAP can beat each of the other schemes. For instance, with a high variance (CV=0.3, the dashed line) and a cache hit rate of 5% (X-axis value), REMAP outperforms ECP6. Similarly, using a smaller variance (CV=0.2, the solid line) and a 52% cache hit rate, REMAP outperforms ECP6. Note that with cache hit rates of 51%, 65%, 69%, REMAP obtains lifetimes equal to SAFER32 using...
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Figure 3.7: Memory lifetime with different cache hit rates when using REMAP.

a perfect cache with CV equal to 0.20, 0.25 and 0.30, respectively. As Figure 3.7 shows, REMAP can benefit more when there is high variance across memory cells. The memory lifetime increases by 1.96X, 2.6X, and 2.8X when the cache is considered perfect, as compared to when no cache is utilized for CVs equal to 0.20, 0.25 and 0.30, respectively.

3.2.3 A Second Layer of Defense

We evaluated some of the key fault-tolerance schemes for PCMs and discussed how REMAP can be modified to outperform other schemes. The limited number of faults that are tolerated by previous schemes creates a challenge in terms of designing reliable memory, particularly when considering future technology nodes. REMAP aims to utilize all the metadata bits for fault-tolerance, which can be used as a second layer of defense, pushing the limits of previous schemes further and increasing memory lifetimes. In other words, the first layer of defense exploits another ECC scheme other than REMAPs (e.g., SECDED, ECP6, SAFER32, or Aegis) to reduce the extra write overhead whenever the number of faults is small. However, if the number of faults in a given block is beyond the fault handling capabilities of the first layer of defense, we switch to our REMAP error recovery scheme to keep the data block alive. Therefore, in the metadata, we need to define
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

another bit that is devoted to determine whether the first or second layer error correction is used for fault recovery.

Figure 3.8 shows the memory lifetime for the different schemes, with and without using REMAP as the second layer of defense. The contribution of REMAP, in terms of the increased lifetime, is added as a stacked bar on top of the other schemes. The impact REMAP has on increasing the memory lifetime is more pronounced at higher levels of variance. Note that adding a second layer of defense to keep a data block alive as long as possible incurs cost and design complexity in the ECC logic. This problem is exacerbated when using REMAP with Partitioning Bit-wise Inversion schemes such as Aegis or SAFER, where the logic needed for error correction is completely different. Nonetheless, employing REMAP with ECP6 (both mechanisms performing “bit replacement”) imposes minimal complexity on the ECC logic. Therefore, we propose using ECP6 as the first layer to reduce the extra write overhead when the number of faults is small, and use REMAP later when the errors increase for each memory block. Using ECP6 with REMAP extends memory lifetimes by 9%, 14% and 21%, as compared to Aegis 17×31 assuming different CV values. Additionally, we see a rather small reduction in memory lifetimes of 3%, 5% and 9% for increasing CV values when comparing ECP6 to Aegis 17×31, where both employ REMAP as a second layer of defense.
3.2.4 Comparing All Approaches

Figure 3.9 shows the lifetime of memory for the three optimized versions of REMAP, as well for ECP6, SAFER32 and Aegis. REMAP(static) and REMAP(dynamic) refer to the schemes where static and dynamic partitionings are used, respectively. We report results for the best number of partitions in the design space, which was obtained in Section 3.2.1.1 and Section 3.2.1.2. In Section 3.2.2 we evaluated REMAP assuming cache hit rates from 0 to 100%. To compare against other schemes, we only consider a cache with moderate (60%) and high (90%) hit rates. We show REMAP with fault location caching as REMAP(Cache) in the figures.

In Figure 3.9 REMAP(SLD) refers to REMAP being used as a second layer of defense. In this evaluation, we assume ECP6 is used as the first layer of protection to filter extra writes when the number of errors in a data block is six or less.

The experimental setup is similar to Section 3.1.2 and, we consider improvements in lifetime while assuming different CV values (0.2, 0.25, and 0.3). The following observations are notable:

i) When the location caching hit rate is high enough (90%) to filter the extra writes in faulty data blocks, lifetimes can be dramatically increased in all the cases. Assuming a 60% cache hit rate, REMAP cannot compete with Aegis in the 20% variance scenario. As the variance increases, assuming an average cache hit rate of 60%, REMAP outperforms Aegis at high variance (CV=0.30).

ii) Dynamic partitioning performs better than static approaches in all the cases, but comes with the added cost of increased design complexity. However, REMAP(SLD) outperforms both static and dynamic partitioning approaches, since extra writes are only required for data blocks with more

![Figure 3.9: Page lifetimes after applying optimizations to REMAP, with CV values of (a) 0.20, (b) 0.25, and (c) 0.30.](image)

(a) CV= 0.20  (b) CV= 0.25  (c) CV= 0.30
than six errors. The efficiency of REMAP with partitioning and REMAP(SLD) are quite similar when the CV is 0.30, but 20% better than Aegis.

iii) As process variation in memory cells increases, the effectiveness of Aegis is reduced, as compared to REMAP. We can see a clear gap between Aegis and REMAP with CV=0.30.

Using Monte Carlo simulation, we showed that the proposed optimizations are effective at reducing extra write overhead, helping to tolerate a larger number of faults and increase memory lifetimes. In the following, we further validate our REMAP approach, utilizing trace-driven simulations. We also consider the cost of the ECC logic to support REMAP, and provide the delay impacts of using REMAP when running various benchmarks.

3.3 Trace-driven lifetime evaluation

Memory lifetime simulation introduces certain challenges, as it is infeasible to simulate the actual operation of memory wear-out since this could take years if running at full speed. This problem is further exacerbated when the size of the simulated memory is large. We began our evaluation using Monte Carlo simulation, making a few simplifying assumptions to perform lifetime simulation in a reasonable amount of time. Much of the prior work discussed in this chapter used Monte Carlo simulation to evaluate their error recovery schemes [9, 28, 59, 66, 72, 73, 84].

When using Monte Carlo simulation, we assume that wear leveling is perfect, and that all the data blocks in a page experience exactly the same number of writes. Real applications typically do not access all of the blocks in the memory, and certainly do not access every data block uniformly. Hence, Monte Carlo simulation only reveals one aspect of lifetime analysis, where writes are uniformly distributed across all memory blocks.

3.3.1 Experimental Setup for Trace-driven Simulation

To reduce the simulation time, we assume the number of writes that each cell can tolerate before a failure follows a normal distribution, with a mean (μ) of 10⁵ (versus 10⁸, which is the value used in our Monte Carlo simulations). The number of simulated memory pages is assumed equal to the memory footprint of an application. If a selected error recovery scheme is not capable of tolerating faults in a given data block, the whole page is dead and is marked unusable by the operating system. The simulation continues until all the pages are dead.
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Table 3.1: Workload characteristics.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Read Access (Million)</th>
<th>#Write Access (Million)</th>
<th>#Unique Pages (Kilo)</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferret</td>
<td>3.37</td>
<td>1.86</td>
<td>25.91</td>
<td>1.81</td>
</tr>
<tr>
<td>Swaptions</td>
<td>2.60</td>
<td>1.27</td>
<td>24.96</td>
<td>2.05</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>1.61</td>
<td>0.68</td>
<td>18.00</td>
<td>2.38</td>
</tr>
<tr>
<td>Fluidanimate</td>
<td>2.25</td>
<td>1.29</td>
<td>32.48</td>
<td>1.75</td>
</tr>
<tr>
<td>Canneal</td>
<td>7.39</td>
<td>3.44</td>
<td>23.18</td>
<td>2.15</td>
</tr>
<tr>
<td>Facesim</td>
<td>3.09</td>
<td>2.12</td>
<td>36.10</td>
<td>1.46</td>
</tr>
<tr>
<td>Freqmine</td>
<td>1.49</td>
<td>0.73</td>
<td>19.23</td>
<td>2.04</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>1.87</td>
<td>0.92</td>
<td>20.79</td>
<td>2.04</td>
</tr>
</tbody>
</table>

We replay the trace of applications repeatedly, similar to prior work [94], tracking the accesses made to each memory block. Logical to physical address mapping is performed to implement a perfect wear-leveling algorithm at the granularity of a page. In addition, as read/write operations are performed at a cache block granularity in PCM, we implement a Line Level Write-Back (LLWB) algorithm [67], modeling fine-grained wear-leveling at a Last Level Cache (LLC) line granularity. Fine-grained wear-leveling improves PCM lifetime, particularly for applications which write to very few blocks in a page. To further reduce the simulation time, the wear-leveling algorithms are evaluated after every 1000 repetition of the trace. Similarly, the status of the PCM memory cells and pages are monitored at the same intervals. Note that the former optimization may introduce small artifacts into the results, but greatly reduces our simulation time. The ideal solution would be to monitor cell status on every write access. However, doing so leads to impractical simulation time.

For lifetime analysis, it is important to use traces that stress the main memory in order to evaluate different aspects of the error recovery scheme. Our traces are extracted from execution of the PARSEC suite [11], and are released with the current version of the USIMM simulator [14]. General information for each benchmark, including the number of reads and writes, the number of unique pages touched, and the read/write mix, is presented in Table 3.1. Each memory trace captures roughly 500-750 million instructions, but is collected for 5 billion executed instructions using the Simpoint [75] methodology [14].
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Figure 3.10: Lifetime across our workloads for different ECCs with CV= 0.20. The results are normalized to the ECP6 scheme.

3.3.2 Lifetime Analysis

The lifetime of different benchmarks under various fault recovery schemes, with a CV=0.2 and CV=0.3, is presented in Figure 3.10 and Figure 3.11 respectively. All of the results are normalized with respect to the ECP6 scheme, which experiences the shortest lifetime of all evaluated schemes. Each workload stresses the memory system differently as compared to the access streams generated using the Monte Carlo simulation. The following observations are obtained from trace-based simulations.

i.) The relative improvement of all the evaluated schemes is more pronounced as the variance in cell lifetime increases. The variance in the mean lifetime of the memory cells has a significant impact on the failure behavior of PCM, since it reflects the effect of process variation in the real world. For the best error recovery scheme, the geometric mean of increased lifetime is 25% with a CV=0.2, while it boosts up to 56% with a CV=0.3.

In addition, using REMAP(static) achieves only a 8% lifetime improvement - higher than Aegis for lower variance, while a 16% improvement is achieved at the higher variance. This observation highlights the need to utilize a robust error protection mechanism, especially as the number of errors in memory increase due to process variation in future technology nodes.

ii) The average lifetime results are consistent with our Monte Carlo results. However, there is a significant variation across different benchmarks. Assuming a variance of 30% in cell lifetime, Monte Carlo simulation shows a 32%, 60%, 47%, and 58% improvement over ECP6 for Aegis, REMAP(SLD), REMAP(Static), and REMAP(Dynamic), respectively. We see a 31%, 49%, 51%, and 56% average improvement when using trace-driven simulation, tracking our previous results. But we need to point out that the difference in lifetime improvement across benchmarks is
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Figure 3.11: Lifetime across our workloads for different ECCs with CV= 0.30. The results are normalized to the ECP6 scheme.

significant. For instance, using REMAP(SLD), we see a 69% lifetime improvement for Facesim, while we see only see a 31% improvement for the Ferret benchmark. The benefits in lifetime is more pronounced in Facesim benchmark when using REMAP, particularly for REMAP(Dynamic) which enjoys a 78% improvement as compared to the baseline, ECP6.

iii) In general, REMAP with all of its variations, outperforms the other rival schemes. In particular, REMAP(Dynamic) achieves the best lifetime numbers as compared to other schemes. Moreover, REMAP is more robust when considering higher variance. As the CV increases to 0.30, the geometric mean of the lifetime improvement grows to 56% for REMAP(Dynamic). Aegis always outperforms SAFER, and improves PCM lifetime by 14% and 31% for 20% and 30% variance, respectively.

iv) Caching fault locations with REMAP shows small improvements over Aegis. Caching helps to avoid the extra writes, but it is not effective enough to compete with other REMAP variations. When the number of faulty cells is small, caching avoids the extra writes, however, with an increasing number of faults, the effectiveness of the cache significantly reduced. For instance, in the Facesim workload, the caching scheme performs poorer than SAFER or Aegis. Nonetheless, the geometric mean obtained over all the benchmarks shows a slight improvement as compared to Aegis.

3.4 Overhead Discussion

3.4.1 Hardware Implementation

Vanilla REMAP. The error correction logic required by REMAP needs to replace faulty bits with the correct values in the metadata. An optimized design would require a path from the
metadata cells to the data cells through intermediate switches. This implementation, which is similar in structure to a complete bipartite graph, is too costly. A more efficient design, in terms of circuit complexity, is shown in Figure 3.12. The decoder has an enable input signal to set all the outputs to zero when there is no error in a block. This logic simply allows any of the data block bits to be replaced by any bits in the metadata. However, the replacement operation is performed sequentially, a bit at a time. To be able to replace $n$ bits concurrently, the same logic should be replicated $n$ times. Therefore, we need to consider the trade-off between circuit complexity and the cycle overhead for the error recovery. It is noteworthy that the access latency of the PCM array is much higher than that of the extra logic needed for error recovery, which results in only a minor impact on the final PCM access latency.

**REMAP Variations.** Any variation of REMAP adds extra hardware overhead to the initial design. ECP and REMAP can share the same hardware logic for error recovery, as both utilize bit replacement. Therefore, when exploiting REMAP as the second layer of defense alongside ECP, there is little added overhead. We achieve a 21% lifetime improvement as compared to Aegis $17 \times 31$
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

Table 3.2: Main characteristics of the simulated system.

<table>
<thead>
<tr>
<th>Processor, On-chip Caches</th>
<th>Processor</th>
<th>1 core, out-of-order, 2.5GHz, Solaris 10 OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 caches</td>
<td>Split I and D, 32KB private, 2-way, 32B, LRU, write-through, 1-cycle hit, MSHR: 4 instruction &amp; 32 data</td>
<td></td>
</tr>
<tr>
<td>L2 cache</td>
<td>256KB, 8-way, 64B, LRU, write-back, 7-cycle hit, MSHR: 32 (I and D)</td>
<td></td>
</tr>
<tr>
<td>Main Memory</td>
<td>Controllers</td>
<td>1 controller, 32×64B write queue, 8×64B read queue.</td>
</tr>
<tr>
<td>Memory</td>
<td>1 channel, 1 DIMM per channel, 1 rank per DIMM, 8×8 chips per rank, 8 banks per chip; PCM cell: 60ns read, 50ns RESET, 120ns SET; Frequency: 400MHz, $t_{RDC}=60$ cycles*, $t_{CL}=5$ cycles, $t_{WL}=4$ cycles, $t_{CCD}=4$ cycles, $t_{WTR}=4$ cycles, $t_{RTP}=3$ cycles, $t_{RP}=60$ cycles, $t_{RRDact}=2$ cycles, $t_{RRDpre}=11$ cycles</td>
<td></td>
</tr>
</tbody>
</table>

* Here, cycle refers to memory cycle at 400MHz frequency.

when REMAP is used with ECP (see Figure 3.8). When REMAP is used with static partitioning, a simple OR tree logic should be added to error recovery logic to determine which partition is faulty. However, dynamic partitioning requires more complex logic, similar to Aegis [28]. Using REMAP with a 4-way set-associative cache having 16K entries incurs only small energy and delay overheads.

We use Cacti [86] to compute the power and access time for the cache configuration assuming 32 nm technology [1]. The ITRS-LSTP (Low Standby Power) values are selected for SRAM cells to reduce the leakage power significantly, since the cache is not used heavily until the memory cells start to wear out. The cache access time is 1.14 ns, consuming 26 mWatt of dynamic power and negligible leakage power. Note that the cache used for storing error locations is not a critical component in terms the correct functioning of REMAP. However, it could be augmented with a low-cost error detection mechanism such as parity. Whenever data in the cache is faulty (a hard or soft fault), the error location could be found by an extra write mechanism used in our vanilla REMAP scheme.

3.4.2 Delay Overhead Analysis

We evaluate the delay overhead of the vanilla REMAP scheme using the cycle-accurate Gem5 simulator [12]. The main memory model is based on using DRAMSim2 [71], which was
modified to include PCM details. The configuration parameters of the simulated system are provided in Table 3.2. Our main memory is designed using SLC-based PCM. The parameters for the PCM are set to model a standard PCM-based DIMM, similar to values used in prior work [43, 44]. We use the PARSEC benchmark suite [11], providing us with a robust set of multi-threaded workloads. Each benchmark is compiled with full optimization and run on the simulated system for one billion instructions, after a one billion instruction phase to get beyond the start-up phase, and to warm up the caches.

In Section 2.3, we discussed the benefits of adding error correction logic on the DIMM, in order to avoid bus contention between the processor and memory system. However, in order to avoid a customized and non-standard design, we assume that the error correction logic is present in the memory controller. We consider the added bus and read/write queue overhead incurred in our simulation.

We evaluate the performance loss, assuming different error rates, ranging from 1% to 10%. We assume the error rate is uniformly distributed among memory accesses. Moreover, we consider REMAP without any of the three enhancement schemes for performance evaluation. In Figure 3.13, we report on the performance loss, as compared to a system with a perfect memory system (i.e., without any errors). Canneal shows the highest performance loss, experiencing an 11% performance loss when 10% of the accesses experience errors. However, for some of the other benchmarks such as Streamcluster and Blacksholes, the performance loss is lower than 4% (for the highest rate of error of 10%). The geometric mean shows that we have only a 0.5%, 3.0%, and 6.2% performance loss, assuming 1%, 5% and 10% of errors, respectively.

The latencies are for the 32 nm technology at 2.5GHz.
CHAPTER 3. REMAP: A RELIABILITY/ENDURANCE MECHANISM FOR ADVANCING PCM

3.5 Summary of REMAP Approach

PCM wear-out continues to be a barrier, impacting the potential of this novel technology to replace DRAM in future main memory. REMAP attempts to boost the reliability of PCM, extending the lifetime by leveraging all the metadata bits as replacements for faulty bits in data. However, REMAP requires an extra memory access in order to locate the faulty bits. To eliminate or reduce this overhead, we consider three optimizations: partitioning, fault location caching, and using REMAP as a second layer of defense. We show that by adopting REMAP, PCM lifetime is drastically improved, handling many faulty bits per each memory block, while not imposing significant performance overhead. We believe that REMAP is an effective design for robust hard-error correction support in PCM. REMAP can enhance PCM lifetime by up to 78% as compared to the baseline ECP6, and up to 24% as compared to the best alternative scheme, Aegis.
Chapter 4

Block Cooperation to Extend Memory Lifetime

When a fault is detected in PCM, an error correction mechanism is required for recovery. Extra memory bits are used as the metadata for the Error-Correcting Code (ECC) scheme to detect and correct the faulty bits, increasing memory lifetimes. Since error profiles in resistive memories are handled completely differently than in DRAM, a number of novel error correction schemes have been proposed to recover from stuck-at faults in resistive memories [9, 28, 59, 66, 72, 73, 91].

However, the proposed error correction schemes for NVMs do not utilize metadata efficiently. In this chapter, we propose block cooperation to solve the metadata under-utilization problem and extend the lifetime beyond what can be obtained by state-of-the-art error recovery schemes. The rest of this chapter is organized as follows.

Section 4.1 provides a simple simulation on error correction schemes in NVMs to highlight the problem of metadata under-utilization. Section 4.2 introduces the concept of block cooperation and shows the state-chart model of our scheme. Section 4.3 describes adoption of the block cooperation into the two state-of-the-art error recovery schemes. Section 4.4.1 describes our experimental framework for both Monte-Carlo and trace-driven simulation, workload characterization and simulation results. Finally, in Section 4.5 we summarize the lessons learned in this chapter.

4.1 Motivation

Although there are several techniques to evenly distribute the number of writes across all the memory cells, given reliability issues with shrinking technology nodes and the associated
manufacturing challenges [94], as well non-uniform write patterns in many applications [65, 67, 84], error correction schemes are less effective. To show the impact of non-uniformity on memory lifetimes, we model more than two thousand 4K pages, each containing 64 memory blocks. We define five scenarios with generating different synthetic workloads and PCM parameters. Each scenario is described as following.

1. We assume each block experiences 50% bit flips in each write, and the flips are uniformly distributed across cells. We also assume perfect wear leveling and uniform writes across all memory blocks. The mean value of cell lifetimes is assumed $10^8$, which follows a normal distribution, with a dispersion of 20% around the mean value (i.e., Coefficient of Variance or CoV = 0.2).

2. This scenario represents the impact of process variation on the lifetime of cells. This scenario is similar to scenario 1, except we increase the coefficient of variance (CoV) to 0.3.

3. Workloads exhibit a different number of bit flips during each memory writeback. To synthetically reproduce this behavior, we randomly flips 10%-50% of cells at each write. The other parameters are similar to scenario 1.

4. Workloads usually do not write to all memory blocks of all the pages. To reproduce this level of non-uniformity in workloads, we select between 1 to 32 blocks to write in each page, versus writing all blocks. The other parameters are similar to scenario 1.

5. In this scenario, we combine all the previous non-uniformity in workloads and process technology. The CoV is assumed to be 0.3, bit flips per writes are 10%-50%, and we only select 1 to 32 blocks out of 64 blocks in each page to be written.

We model Error Correction Pointer (ECP) [72] and Aegis [28], two state-of-the-art error correction schemes proposed for resistive memories. Once an error correction scheme can no longer tolerate faults in a data block, the entire physical page associated with that data block must be disabled, and becomes unavailable to the physical address space. We continuously write to the memory and record the number of errors in each data block until all the pages are taken offline. In Figure 4.1, we plot the memory lifetime in terms of billion writes per page (Y-axis) versus the average number of errors per page (X-axis), for both ECP and Aegis. Each point represents a specific scenario for synthetic workloads.

\[\text{We describe our simulation methodology in detail in Section 4.4.1}\]
Figure 4.1: Memory lifetime and the average error per page for different scenarios. Each scenario mimics different behavior of workloads or memory system. Non-uniformity in write accesses reduces PCM memory lifetime significantly and results in under-utilization of error correcting chip for error recovery.

By stressing the memory with different scenarios, we make the following observations:

1. non-uniformity in write accesses, as well as the manufacturing process, can dramatically reduce the memory lifetime. Note that the non-uniformity increases gradually from scenario 1 to scenario 5. Using the number of writes per page (Y-axis) as a proxy for memory lifetime, we can impact wearout by a factor of 8X and 6X for ECP and Aegis, respectively, in the worst-case.

2. Even though ECP can tolerate up to six faults per data block, almost 77% of data blocks experience three or fewer faults in scenario 1. On the other hand, Aegis\textsuperscript{2} can tolerate 10 faults (deterministically), and is able to tolerate more faults (probabilistically). However, almost 50% of all of the data blocks tolerate five or fewer faults. Increasing non-uniformity severely reduces the effectiveness of using error correcting codes. We see a shift to the left when compare the average number of errors per page occurring, as we compare scenarios 1 through 5. The average number of corrected errors per page is reduced by a factor of 4X for both error correcting schemes when experiencing increased non-uniformity.

\textsuperscript{2}The configuration for Aegis is considered $17 \times 31$ \cite{28}.

42
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

The observations for these two different error recovery schemes confirms that non-uniformity hampers the effectiveness of error correction, and reduces the utility of the ECC bits, significantly. Informed by these observations, we propose using block cooperation, a simple but effective technique that can be incorporated with different error correction schemes to boost PCM lifetimes, while imposing little extra cost.

To evaluate the effectiveness of our proposed approach, we exploit block cooperation on top of ECP [72], and Aegis [28] error correction schemes. Block cooperation allows memory blocks that experience a small number of errors to come to the rescue of blocks with a larger number of errors, avoiding an early page death due to non-uniform writes. Block cooperation is realized through metadata sharing in ECP. Metadata sharing can be performed at a single level, where one data block shares its unused metadata with another data block, or can be implemented to be multi-level, where multiple data blocks can share their metadata together. In Aegis, block cooperation is realized through data layout reorganization, where the blocks with fewer faults can help failed blocks in order to bring a page back to life. Using single level (or multi-level) block cooperation, we can increase memory lifetimes by 28% (37%), and 8% (14%) on average, for ECP and Aegis, respectively.

4.2 Block Cooperation

Applications do not access all data blocks uniformly. The write endurance of cells within a data block is also not equal across all cells. Some will wear out sooner than others (surviving fewer writes). When ECC is no longer able to tolerate faults in a data block, the operating system will remove the entire page that contains that block. Hence, the weakest block in a page dictates the fate of the rest of the block in a page. This is while the metadata of the other blocks within the page remain under-utilized, and while the other blocks in the page may experience no or only a few faults. Block cooperation techniques address this problem by allowing data blocks to share their data and/or metadata cooperatively, extending memory lifetime.

In general, block cooperation can be performed on a single level, where only a single live block cooperates with another dying block, or can be performed across multiple levels, where multiple live blocks are able to cooperate with a single dying block. A control field is required in each block’s metadata to maintain the state of the data blocks and any information that supports cooperation. An indirection pointer is used to help the memory controller quickly find a cooperating block or blocks.
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

To initiate the cooperation procedure, the memory controller needs to be able to find a candidate block. The selection policy can be done randomly or can follow a set of policies. To record each state, and manage transitions through these states during the lifetime of a data block, we use statecharts. Depending on the state of a data block, the memory controller is responsible for updating the block state and initiating any required operations to handle block cooperation. Statecharts are more flexible than finite-state machines [33]. Figure 4.2 shows a statechart that describes our block cooperation algorithm. Note that the transitions presented by dotted arrows only occur during multi-level cooperation. Figure 4.2 also shows the binary representation of each state. For every transition, the control field of the metadata is updated by this binary representation.

At the highest level of the statechart hierarchy, the data block status contains two super-states, faulty and non-faulty. Once a cell becomes faulty due to wear-out, the state for the data block transitions from non-faulty to faulty. Within these two superstates, the following sub-states are defined:

- **Private.** The block does not cooperate with other blocks.
- **Shared.** The block cooperates and shares its data or metadata with another block.
- **Shared+.** The block shares its data or metadata with another block, and uses indirection pointer to refer to another shared or shared+ block.
- **Co-op.** The block is dead and has requested cooperation to become live again.

The following actions/operations are also defined to facilitate our cooperative scheme:

- **Join.** When a block needs to cooperate, a candidate block is selected, and then a join operation is initiated. The candidate block needs to be in the private state, and after joining, the state of the block changes to shared.

- **Disjoin.** A block in the shared or shared+ state may not share its data or metadata for its own benefit. In this case, a disjoin operation is performed, and a transition to the private state is made. Since the corresponding block in the Co-op state has lost its block cooperator, a different data block(s) needs to be selected to join and save the entire page from failure.

- **Indirection.** When a block needs to cooperate with another block to survive, it transitions to the Co-op state, and the cooperator block transitions to the shared state. In case of multi-level cooperation, a data block may request cooperation with more blocks. Hence, an indirection
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Figure 4.2: Block cooperation statechart. At the highest level of the statechart hierarchy, the state of blocks is either faulty or non-faulty. Each of these superstates contains various states, which transitions to a new state based on the specific operation.

operation is performed to have another block join the shared pool. This operation also sets an indirection pointer for the cooperating block. By adding indirection pointers to the metadata, the block can transition to the shared+.

- **Cooperation request.** Requests for cooperation are initiated when i) a data block is in the private state and is about to die, or ii) during a disjoin or indirection operation, where a data block requests cooperation with more blocks. If the block selection policy finds a block which can resuscitate the failed block, the data block transitions to the Co-op state (if it is not already in that state). Otherwise, it is no longer possible to save the block and the entire page must be disabled.

For block cooperation, the state-chart can be simply implemented using a finite-state machine and reserving three bits of metadata in the ECC chip.

4.3 Block Cooperation Adoption

To show the effectiveness of our approach, we select two error correction schemes designed specifically for resistive memories: i) ECP [72], and ii) Aegis [28]. We integrate block cooperation with both of them. Next, we outline the range of design parameters considered in this work.

Block cooperation is integrated with ECP and Aegis using metadata sharing and data layout reorganization techniques, respectively. In the former, data blocks cooperate by sharing their
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

metadata, while in the latter, blocks cooperate through data sharing and shuffling. In the following, we describe the details of block cooperation adoption.

4.3.1 Block Cooperation in ECP

The Error Correction Pointer (ECP) [72] is the pioneering work for using ECC in resistive memories. The faulty bits are located by a read verification after the write. Then for every faulty bit, one pointer and one replacement bit are kept in the metadata (forming one ECP entry). The metadata storage requirement increases by introducing more faults in a data block. Error correction is performed by replacing the correct bit with the bit pointed to by the pointer. In a standard DIMM with one ECC chip per eight data chip, 8-byte of metadata are available for a 64-byte data block, ECP provides the ability to correct up to six faults, which we will refer to as ECP6 throughout the rest of this chapter.

In the ECP6 scheme, one bit is used to indicate whether a block is faulty or not, and 60 bits are used to keep track of the six ECP entries. As discussed in Section 4.2, three bits are required to maintain a block’s status and enable metadata sharing for ECP. When a block is in the private state, the rest of the metadata (61 bits) provides six ECP entries. Therefore, the error correction capability (in the worst case) is not less than the standard ECP6.

Figure 4.3 shows the metadata format when single or multi-level sharing is used. When a block is in the Co-op state, six more bits are required for the indirection pointer. The indirection pointer field shows the location of helper block in page. Five bits indicate the number of extra ECP entries required by the Co-op block, thereby providing the capability of tolerating 36 faults for a block (in the best case). Providing block cooperation requires additional fields in the metadata, which results in reducing ECP entries to 5. In other words, we can only have 6 ECP entries if a block is in the private state, otherwise the number is reduced to 5.

When a block is in the shared or shared* state, three bits are required to track the number of ECP entries used. This field is essential for distinguishing between the ECPs that are used privately for a block, and the ECPs that are shared with other blocks. In the disjoin operation, the state of the current block, and the indirection pointer of a block that points to the current block need to be updated. In the shared state, six bits are allocated to indicate the borrower block. This field is not essential for the functionality of metadata sharing. However, the field provides a structure, similar to a circular list, that is used during the disjoin operation. In this way, the helped block can easily be reach in order to join it with another helper block.
The policy for selecting a block for join operations can be simply a random policy. However, the better selection policy is to find a block with no or the lowest number of faulty cells. To this end, all of the metadata in the block should be read, which incurs one (best-case) to 63 (worst-case) read(s) in the corresponding page. Note that this operation is not frequent, and aggregation of extra read latencies in the case of a row buffer hit in the page is small [37].

4.3.2 Block Cooperation in Aegis

Aegis [28], which similar to SAFER [73] and RDIS [57], uses partitioning and bit inversion to mask stuck-at errors by storing faulty values in their inverted form. To exploit fault masking in these error correction schemes, faults should be isolated logically using partitioning. The goal of the partitioning scheme is to guarantee the existence of no two faults in the same partition.

Aegis partitioning distributes faults into different groups. When a new error is introduced in a group with an error, re-partitioning is performed to resolve error collisions. Aegis performs this partitioning by mapping data blocks to a 2-D Cartesian plane with $A \times B$ elements. On a Cartesian plane, any two different points on a line determines the slope of the line. Changing the slopes of the line preserves, at most, one cell of the original line to stay on a new line. All the cells that are
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

covered by a line, belong to the same group. In the Aegis error correction scheme, there are B lines that share a common slope, so each cell can be covered by only one line or group. The re-partitioning is performed simply by changing the slope associated with a block. The slope can be stored with only \( \lceil \log_2 B \rceil \) bits.

The plane for a simplified 32-bit data block, in a 5 × 7 organization, is shown in Figure 4.4a. In this format, each error is shown as a black square on the plane, while partitions are the B parallel lines (in this example B is equal to 7). The slope of the parallel lines should be chosen such that no two faults are placed on the same line or partition. If a valid slope (i.e., \( 0 \leq K < B \)) is not found that can satisfy this property in a block, error recovery is not possible.

To describe how block cooperation is adopted in Aegis with our simplified Cartesian plane, we show three different blocks in Figure 4.4a. Blocks \( \alpha \), \( \beta \), and \( \gamma \) contain six, four and one faults, respectively. No valid slope is found for block \( \alpha \), whereas blocks \( \beta \) and \( \gamma \) can tolerate the fault pattern with slopes of one and zero, respectively.

Similar to ECP, we can exploit block cooperation, allowing multiple blocks to work cooperatively to keep their associated page alive. In contrast to ECP, Aegis metadata is fixed and independent of the number of faults in a data block. Therefore, it is not possible to share metadata in order to exploit underutilized metadata, so a different approach is needed for block cooperation.

To extend the memory lifetimes in Aegis, we use data layout reorganization, where blocks with fewer faults help dying blocks (i.e., blocks with no valid slope) to keep the page alive. For instance, in Figure 4.4b three blocks are joined together and each portion of the plane (i.e., \( P_1 \), \( P_2 \) and \( P_3 \) in Figure 4.4b) is filled by one of the three blocks. The new slopes for new data layout are calculated to check whether all the faults in the data blocks are recoverable. Note that the data layout reorganization is performed in the Aegis buffer in the memory controller. A block that requests cooperation transitions to the Co-op state, and the block that is selected to join transitions to the shared state. If we fail to find a valid slope for the new data layout, more blocks are gradually invited join the Co-op block to help. Block cooperation in Aegis helps to spread faults across different blocks more evenly, resulting in limited page waste due to early block failures. For example, in Figure 4.4a the number of faults are six, four, and one in the data blocks that change to four, four, and three in Figure 4.4b after applying data layout reorganization.

Assuming two blocks cooperate to save a faulty block, but no valid slope \( K \) is found for at least one of them, then the controller needs to select another private data block to perform the join operation. The number of retries that can be performed is equal to the number of private blocks within a page. We can also select a predefined maximum try threshold. For instance, the block \( \alpha \) in
(a) Aegis with no block cooperation. No valid slope is found for block $\alpha$, which results in disabling an entire page. With a slope equal to 2, four faulty cells can reside in four different partitions in the block $\beta$. For block $\gamma$, a slope of 0 is enough to tolerate one fault.

(b) Aegis with multiple block cooperation. Blocks $\beta$ and $\gamma$ are joined with the dead block $\alpha$ to save the entire page. Data layout reorganization is able to find a valid slope, in order to revive the dead block. Each portion of the plane (i.e., $P_1$, $P_2$ and $P_3$) is filled by one of the three blocks.

Figure 4.4: Integrating block cooperation in Aegis.
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Figure 4.5: The format of the metadata for Aegis \((17 \times 31)\) with block cooperation.

Figure 4.4a cannot be saved, even if cooperates with the block \(\beta\). However, for the next retry, if the block \(\gamma\) is selected, the block \(\alpha\) can be revived.

In multi-block cooperation, new blocks can be joined with a Co-op block through indirection, until there are no more private blocks. Alternatively, a predefined limit can be set for the number of shared blocks, in order to manage the complexity of error correction. We define this variable as the sharing level in the data layout reorganization. Note that in metadata sharing in ECP, accesses to Co-op blocks requires accesses to helper blocks for recovery, but helper blocks are self-contained and accessing them does not incur extra accesses to other blocks. In data layout reorganization, this is not the case, because for data recovery for both helper and helped blocks, issuing accesses to other blocks is necessary.

Figure 4.5 breaks down the required fields for the Aegis metadata when block cooperation is enabled. Just as with ECP, Aegis requires both a 3-bit Block status field and an indirection pointer field. Additionally, a \(\lceil \log_2 B \rceil\)-bit field is required to keep track of the slope of the lines, and \(B\)-bits are required to save the inverse indicator for partitions. The inverse indicator bits denote whether the data is stored in its actual or inverted form.

4.4 Simulation Results and Workload Characterization

4.4.1 Monte-Carlo Simulation for Memory Lifetime Analysis

We begin our evaluation using Monte-Carlo simulation, making a few simplifying assumptions to perform lifetime simulation in a reasonable amount of time. Much of the prior work discussed in this thesis used Monte-Carlo simulation to evaluate their error recovery schemes \([9,16,28,59,66,72,73,83,84]\).
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

4.4.1.1 Experimental Setup

To evaluate the effectiveness of block cooperation, we exploit Monte-Carlo simulation. We model a PCM with 2,048 4K-pages, with 64 memory blocks per page. The memory lifetime is assumed to follow a normal distribution, with a mean of $10^8$. Coefficient of variation (CoV) values of 0.2, and 0.3 are used in our experiments to model different degrees of imperfection in process technology [72, 83, 94]. The higher the CoV, the higher the variability of the lifetimes across memory cells. For each 512 bits of data, 64 bits of metadata is maintained, as is used in standard ECC-based DIMM memories. We also consider the impact of wear-out on the metadata in our simulations.

The probability of a bit flip in a cell is assumed to be 0.5 throughout the simulations. Similar to the previous work [28, 59, 72, 73, 83], we assume perfect wear leveling across the memory blocks. The entire page is disabled if ECC is not able to correct faults in a memory block, and the write in that block is uniformly distributed among other blocks.

4.4.1.2 Lifetime Results

The memory lifetimes, when single and multi-level metadata sharing are integrated with ECP, are presented in Figure 4.6. For single-level metadata, as the CoV parameter is varied from 0.2 to 0.3, lifetime improves over standard ECP6 from 9% to 28%. Using multi-level sharing boosts the

Figure 4.6: Lifetimes using standard ECP6, single level metadata sharing, and multiple level metadata sharing, with CoV values of (a) 0.20, and (b) 0.30.

![Graph showing lifetime improvements with different CoV values and metadata sharing methods.](image-url)
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Figure 4.7: Lifetimes using standard Aegis, Aegis with single sharing level, and Aegis with multiple sharing level, with CoV values of (a) 0.20, and (b) 0.30.

lifetime improvement to 12% and 37%, with CoV values of 0.2 and 0.3, respectively. As Figure 4.6 shows, higher CoV results in better lifetime improvements when using metadata sharing (differences between ECP6 and sharing increase from left to right). Therefore, when using technology with high process variation, exploiting a metadata sharing technique with ECP6 is more effective and will increase lifetimes considerably.

The policy for selecting a block for join operations is to find a block with no or the minimum number of faulty cells within the page. We also consider no limitation on the number of blocks for multiple cooperation.

Figure 4.7 shows page lifetimes with single and multiple levels of sharing integrated with Aegis, and compares against Aegis without any cooperative sharing. CoV values of 0.20, and 0.30 are again considered. For single sharing, the maximum retry threshold is set to 4, while this parameter is increased to 8 for multi-sharing. We choose to increase this limit so that we can explore the limits of cooperation in our evaluation. The policy for selecting a block for join operations is random, as there is no notation of the number of faults in Aegis metadata as compared to ECP. The memory lifetime increases by more than 3% (6%), and 8% (14%) when single (multiple) sharing is used. Just as when we added block cooperation to ECP, Aegis also shows better results with higher CoV.
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Table 4.1: The system specifications of our simulated system.

<table>
<thead>
<tr>
<th>Processor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>X86</td>
</tr>
<tr>
<td>CMP</td>
<td>4-core, out-of-order, 2.66GHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interconnect Network</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-chip Cache Hierarchy</td>
<td></td>
</tr>
<tr>
<td>L1 I-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>256KB/64B/8 way, private</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>8MB/64B/16 way, shared</td>
</tr>
<tr>
<td>Store Policy</td>
<td>Write-Back</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main Memory Configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Capacity</td>
<td>8GB (4KB pages, and 64B rows)</td>
</tr>
<tr>
<td>Configuration</td>
<td>4 memory controller</td>
</tr>
<tr>
<td></td>
<td>1 rank/DIMM, 8 devices/rank,</td>
</tr>
<tr>
<td></td>
<td>1 chip for ECC, ECP [72] and Aegis [28] error corrections</td>
</tr>
</tbody>
</table>

4.4.2 Trace-Driven Simulation for Memory Lifetime Analysis

We use SniperSim [13] to execute workloads and generate memory traces of applications from the SPEC2006 benchmark suite [77]. Table 4.1 describes the parameters used in the modeled system. Our baseline is a 4-core out-of-order multiprocessor which includes three levels of on-chip cache in the memory hierarchy. The benchmarks are executed in rate mode [60], where all the four cores execute the same benchmark.

Memory write requests are captured at the PCM memory, and on evictions from the last level cache (i.e., L3 in our simulations). The granularity of read and write requests are 64 bytes (i.e., one cache line), and are serviced by one of the PCM banks. The first two billion instructions for each benchmark are used for warming up on-chip caches. All references to main memory, each 64 bytes of data, are captured. We terminate trace recording after either 10 million main memory references, or 20 billion executed instructions, whichever occurs first. Note that 10 million references typically correspond to several billions of instruction executed for each benchmark.

4.4.2.1 NVM Workload Characterization

One of the most valuable insights derived from workload characterization is the identification of key architectural features that will dominate program execution. Past workload characterization
has focused on performance optimization \([11,22,34]\). However, in this chapter, we identifying quantitative metrics for a workload that will most impact non-volatile memory. By providing a set of quantitative workload metrics, we can better understand the relationship between workload and memory lifetimes. We perform characterization using detailed trace-driven simulation that models actual data values transferred between on-chip memory and main memory. In Figure 4.8, we present an overview of the various steps comprising our framework, which are used for endurance analysis and associated workload characterization.

We study 13 different benchmarks, providing us with a range of different write intensities and behaviors. The percent of bit flips per block on a writeback varies from 5% to 45% across the benchmark suite \(3\). For all of the benchmarks we used the reference inputs. It is noteworthy that the input of a benchmark can change the size of the memory footprint, and even the read/write characteristics of an application. However, as our main goal is to investigate the relationship between memory writeback characteristics and memory lifetime, we did not evaluate program behavior with multiple inputs.

The following metrics were captured across all the benchmarks. Results are provided in Table 4.2 and Table 4.3:

- **Average % of Bit Flips per Writeback to Blocks.** Applications only modify a limited portion of

\[\text{Writing 100 bits to a memory block is approximately equal to } 100/512 \approx 20\%.\]
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Table 4.2: Collected Metrics Related to Bit Flips at Block and Page level for Different Benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Avg. % of Bit Flip (Block)</th>
<th>Avg. Kilo Bit Flip (Page)</th>
<th>Block-level Bit Flip Intensity</th>
<th>Page-level Bit Flip Intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>astar</td>
<td>5.7</td>
<td>1.3</td>
<td>577</td>
<td>22.477</td>
</tr>
<tr>
<td>calculix</td>
<td>43.0</td>
<td>23.6</td>
<td>1419</td>
<td>85.664</td>
</tr>
<tr>
<td>gobmk</td>
<td>13.7</td>
<td>26.0</td>
<td>1439</td>
<td>81.632</td>
</tr>
<tr>
<td>gromacs</td>
<td>26.3</td>
<td>45.6</td>
<td>2849</td>
<td>154.865</td>
</tr>
<tr>
<td>hmmer</td>
<td>28.1</td>
<td>51.7</td>
<td>2542</td>
<td>155.31</td>
</tr>
<tr>
<td>lbm</td>
<td>4.9</td>
<td>1.2</td>
<td>280</td>
<td>17.02</td>
</tr>
<tr>
<td>leslie3d</td>
<td>28.1</td>
<td>22.9</td>
<td>1171</td>
<td>69.655</td>
</tr>
<tr>
<td>libquantum</td>
<td>4.4</td>
<td>10.8</td>
<td>262</td>
<td>16.004</td>
</tr>
<tr>
<td>omnetpp</td>
<td>11.8</td>
<td>4.8</td>
<td>749</td>
<td>20.948</td>
</tr>
<tr>
<td>perfbench</td>
<td>22.7</td>
<td>7.3</td>
<td>953</td>
<td>44.226</td>
</tr>
<tr>
<td>sjeng</td>
<td>9.1</td>
<td>4.4</td>
<td>602</td>
<td>23.783</td>
</tr>
<tr>
<td>soplex</td>
<td>15.9</td>
<td>8.5</td>
<td>943</td>
<td>58.525</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>16.0</td>
<td>6.4</td>
<td>1339</td>
<td>63.915</td>
</tr>
</tbody>
</table>

the 64 byte data block on each memory writeback. As a PCM chip only writes the differences using the DCW mechanism, a limited number of bit flips will be exposed. The higher the number of bit flips, the more lifetime degradation experienced in the PCM. lbm, astar and libquantum benchmarks only experience, on average, around 5% bit flips on each writeback, while for the calculix benchmark, the percentage grows to 43%. The average percentages of bit flips per writeback across the studied benchmarks is 17.7%.

- **Average Kilo Bit Flip for Pages.** This metric captures the number of bit flips at a page level granularity. hmmer and lbm exhibit 51.7 and 1.2 Kilo bit flips per page, on average, which are the two ends of spectrum, respectively. The average for this metric is 16.5 Kilo bit flips across all benchmarks.

- **Block-level Bit Flip Intensity.** If one of the memory blocks within a page experiences more errors than the tolerable limit, the operating system will discard entire the page. Therefore, a single memory block out of the 64 blocks in a page can dictate the entire page’s lifetime. In other words, approximately 1/64≈2% of the blocks that experience the highest number of bit flips contribute the most to page wear-out, and reduce memory lifetime. The Block-level Bit Flip Intensity metric is computed as the average number of bit flips per block, for the 2% of the blocks that experience the highest number of bit flips. A well-designed wear-leveling algorithm should effectively distribute these bit flips across all the blocks in memory.

- **Page-level Bit Flip Intensity.** In case of a page failure, the operating system needs to remap
Table 4.3: Collected Metrics Related to Read/Write Accesses of Different Benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Writes (Million)</th>
<th>#Reads (Million)</th>
<th>#Written Page (Thousand)</th>
<th>Avg. Writes per Page</th>
<th>Spare Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>astar</td>
<td>4.09</td>
<td>5.91</td>
<td>72.49</td>
<td>56.4</td>
<td>27.6</td>
</tr>
<tr>
<td>calculix</td>
<td>2.77</td>
<td>7.23</td>
<td>23.50</td>
<td>117.8</td>
<td>85.1</td>
</tr>
<tr>
<td>gobmk</td>
<td>4.83</td>
<td>5.17</td>
<td>6.33</td>
<td>763.1</td>
<td>316.0</td>
</tr>
<tr>
<td>gromacs</td>
<td>0.44</td>
<td>0.83</td>
<td>1.29</td>
<td>338.6</td>
<td>1552.8</td>
</tr>
<tr>
<td>hmmer</td>
<td>2.79</td>
<td>2.85</td>
<td>7.78</td>
<td>358.8</td>
<td>257.0</td>
</tr>
<tr>
<td>lbm</td>
<td>3.35</td>
<td>6.65</td>
<td>52.36</td>
<td>64.0</td>
<td>38.2</td>
</tr>
<tr>
<td>Leslie3d</td>
<td>2.57</td>
<td>7.43</td>
<td>13.52</td>
<td>190.1</td>
<td>147.9</td>
</tr>
<tr>
<td>libquantum</td>
<td>3.44</td>
<td>6.56</td>
<td>8.20</td>
<td>419.9</td>
<td>243.9</td>
</tr>
<tr>
<td>omnetpp</td>
<td>3.77</td>
<td>6.23</td>
<td>41.57</td>
<td>90.8</td>
<td>48.1</td>
</tr>
<tr>
<td>perlbench</td>
<td>2.08</td>
<td>3.91</td>
<td>30.97</td>
<td>67.3</td>
<td>64.6</td>
</tr>
<tr>
<td>sjeng</td>
<td>4.47</td>
<td>4.53</td>
<td>44.03</td>
<td>101.6</td>
<td>45.4</td>
</tr>
<tr>
<td>soplex</td>
<td>3.69</td>
<td>6.31</td>
<td>22.43</td>
<td>164.6</td>
<td>89.2</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>1.98</td>
<td>8.02</td>
<td>25.56</td>
<td>77.4</td>
<td>78.3</td>
</tr>
</tbody>
</table>

the page to another location in memory. This gradually reduces the spare blocks available and shrinks the size of the memory system. Consequently, the pressure the on memory system escalates over time and memory lifetime decreases. The Page-level Flip Intensity metric is computed as the average number of kilo-bit flips (i.e., 1000 bit flips) in a page, for the 2% of pages that experience the highest number of bit flips. We expect that, in the future, these pages will cause more wear-out.

- **Number of Writes (Reads).** This metric captures the number of write (read) accesses, that are captured when recording the trace of each benchmark (a trace is either 10M references to main memory, or 20B instructions, whichever comes first).

- **Written Pages.** This metric records the number of pages that experienced at least one block write.

- **Average Writes per Page.** This metric reports the average number of writes to each page.

- **Spare Factor.** Having a larger capacity memory system, while running an application, provides more spare pages that can come to the rescue during wear-leveling and error correction to prolong memory lifetimes. The spare factor includes the capacity of the whole PCM DIMM, over the capacity of written pages (i.e., a value greater than 1, where higher is better).

Although simplified metrics are good indicators to distinguish workload behaviors, the distribution of writes or bit flips in individual benchmarks cannot be easily captured in a single metric.
Figure 4.9: Distribution of (a) writes, and (b) bit flips at block level for different benchmarks. The red dot shows the median of the distribution.

To this end, we use a Violin plot to show the shape of the probability density of writes and bit flips, at both a block-level and page-level granularity.

Figure 4.9 (a) and (b) present the number of writes and bit flips encountered, at a block level, across the 13 benchmarks. Wider sections of the Violin plot represent a higher density of the points for the given value, while the skinnier sections represent a lower density.

Some of the benchmarks are less variable than the others. For instance, astar, lbm, libquantum, and soplex show much less variation than xalancbmk, perlbench, and gromacs. The number of writes to a single block can reach 972, 177, and 90 for these three benchmarks, respectively. For xalancbmk, the Violin shape is tall and narrow, which means the points are spread across a wide range of values, even though the number of bit flips at a page level can reach 78.1 kilo-bit flips.

Figure 4.10 shows the distribution of writes and bit flips at a page level granularity. In contrast to the block level granularity, the shapes of writes and bit flips are similar in many of the benchmarks. But aggregating behavior at a page level, the similarity decreases. The number of writes per page can reach to 6422 for xalancbmk, while lbm only experiences one write per block in some pages (at most 64 writes per page). xalancbmk experiences a maximum of 572 Kilo bit flips in a page, which is the highest number of bit flips within a page for the workloads studied. For the same workload, the mean number of bit flips across all the pages is only 6.4 Kilo. lbm has a median
Figure 4.10: Distribution of (a) writes, and (b) bit flips at a page level for different benchmarks. The red dot shows the median of the distribution.

value of 26 bit flips per page, and a mean of 1.2 Kilo bit flips per page, and has the smallest total number of bit flips.

4.4.2.2 Experimental Setup

Memory lifetime simulation involves a few challenges, as it is challenging to simulate memory with actual data values to accurately assess memory wear-out for long-running programs. This challenge is further exacerbated when the size of the simulated memory is large [83]. If we adopt Monte-Carlo simulation, we would generally assume that the wear-leveling algorithm is perfect, that all of the blocks in a page experience exactly the same number of writes, and that the probability of a bit flip is 50% across all blocks. However, real applications rarely access all of the blocks in the memory, and certainly do not access every data block uniformly.

In trace-driven simulation, applications can touch a large number of pages. Since memory reliability simulation needs to be carried out at the granularity of a bit level, simulating a large number of pages results in a dramatically larger memory footprint for the simulation, reaching many gigabytes in size. To reduce simulation time, while not sacrificing accuracy in our results, we assume the number of writes that each cell can tolerate before a failure follows a normal distribution, with a mean (µ) of $10^5$ (versus $10^8$, which is the value used in our Monte-Carlo simulations). The number of simulated memory pages is assumed equal to the memory footprint of an application. However
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Figure 4.11: Y axis (petabyte writes) shows the possible amount of writes before half of the memory capacity becomes unavailable. CoV is equal to 0.2.

we scaled the final results with respect to the total size of the main memory to make results across benchmarks comparable and meaningful. Note that the former optimizations may introduce small artifacts into the results, but greatly reduce our simulation time.

We replay the trace of applications repeatedly, tracking the accesses made to each memory block. Moreover, we keep track of modifications at a bit-level granularity to check whether the number of writes to a bit address exceeds wear-out limits or not. Logical to physical address mapping is performed, along with applying a start-gap wear-level mechanism [65]. If a selected error recovery scheme is not capable of tolerating faults in a given data block, the whole page is dead and is marked unusable by the operating system. Our framework maps the address of the dead page to another available physical page. The simulation continues until half of the physical pages in our simulation are dead.

4.4.2.3 Lifetime Analysis

For lifetime analysis, we compare the volume of writes the occur before half of the availability memory pages die, across the four error recovery schemes: ECP6, ECP6 with multiple block cooperation, Aegis $17 \times 31$, and Aegis $17 \times 31$ with multiple block cooperation. We measure write volume in petabytes, and use this metric to evaluate memory durability for each benchmark. This metric helps to identify common memory access behaviors across workloads that impact memory durability, and allows us to compare and evaluate different lifetime enhancement strategies with block cooperation. Figure 4.11 and Figure 4.12 show the number of petabyte writes, across different error recovery schemes, and with a CoV of 0.2 and 0.3, respectively.

Block cooperation significantly increases memory durability in perlbench for both ECP
CHAPTER 4. BLOCK COOPERATION TO EXTEND MEMORY LIFETIME

Figure 4.12: Y axis (petabyte writes) shows the possible amount of writes before half of the memory capacity become unavailable. CoV is equal to 0.3.

and Aegis schemes. For instance, we see an increase of 60% (68%) and 18% (30%) in terms of longer lifetimes achieved, with a CoV of 0.2 (0.3) for ECP and Aegis, respectively. However, the average improvement over all the benchmarks is 18% (24%) and 9% (19%) when block cooperation is used with ECP and Aegis, with a CoV of 0.2 (0.3), respectively.

As Figure 4.11 and Figure 4.12 show, we see that memory durability is impacted more in gromacs, calculix, hmmer, leslie3d, and xalancbmk than in the other benchmarks. These workloads generally have higher block-level and page-level bit flip intensities as compared to the other benchmarks. Increasing the CoV from 0.2 to 0.3 reduces the memory lifetime by 31%, on average.

lbm achieves the longest memory lifetime among all benchmarks, which is tied to the memory characteristics of this benchmark. In Figure 4.9 and Figure 4.10, which show the distribution of writes to blocks and pages, lbm has the lowest values. lbm has the smallest number of bit flips per page – the average number of bit flips per block writeback is very small (i.e., 4.9%). The number of written pages, and the number of writes, is large. However, only one write is performed to each block in a page for lbm. Note that metadata sharing in ECP can only increase lifetime by 3%~4% for lbm, while using data layout reorganization in Aegis, lifetime can increase by 25%.

The correlation between block-level bit flip intensity, with the average number of bit flips per block, is moderately positive (0.66), while the correlation between page-level bit flip intensity with the average bit flip per page is strongly positive (0.95). The lifetime correlation with bit flips at a block level and page level is moderately negative, and varies between -0.44~0.54. Benchmarks with a higher number of pages accessed, such as lbm, astart, and omnetpp, experience longer lifetimes, as there is a greater chance to balance wear-out across more pages. Note that there is
moderately strong correlation (almost 0.50) between the number of written pages and the benchmark lifetime.

Since block cooperation more effectively uses the metadata, it can tolerate more errors, thereby improving memory durability. Figure 4.13 shows the petabyte writes occurring (as a representative metric of memory durability) for four benchmarks, each possessing a different number of tolerable errors per page.

The results are for different error correction schemes with and without block cooperation, and vary the CoV. Aegis can correct a larger number of errors, and always produces more correctable errors as compared to ECP. Block cooperation increases the number of correctable errors, although its effectiveness varies with respect to the error correction scheme and the specific benchmark considered. Since ECP can only tolerate 6 errors, the metadata is always underutilized. Multi-block cooperation shows significant potential, across all of the benchmarks, in tolerating more errors and increasing memory lifetimes. For omnetpp using Aegis, block cooperation provides a significant improvement, while xalancbmk sees little improvement.

Exploiting metadata sharing for ECP, and data layout reorganization for Aegis, we can improve memory lifetimes considerably. However, block cooperation is similar to other error correction schemes in that it adds some complexity to the memory system.

It might seem that accessing cooperative blocks introduces some performance degradation, as single or multiple accesses are required for error recovery. Nonetheless, the row buffer in the main memory architecture overfetches data, e.g., 4KB is read for a 64B request. This will impact the latency associated with subsequent memory accesses within the page (i.e., a row buffer hit is around 20 ns [37]). In the absence of block cooperation, the entire page should be disabled, which then would lead to an access to the next level of the memory hierarchy, or a page swap by the operating system. Either of these approaches results in a high latency access (on the order of thousands of cycles). Therefore, saving blocks from failure, at the added cost of a small latency, is much more palatable, especially versus disabling the page.

Page level paring schemes such as DRM [35] and Zombie [9] schemes can be used as a second layer of defense with block cooperation to further increase memory lifetimes. However, they require access to an additional page in memory, which is costly in terms of performance and energy consumption, as compared to block cooperation, which tries to increase correctable errors by exploiting resources within a page.

In ECP, when using metadata sharing, read/write operations from/to only the Co-op blocks impose extra read/write operations from/to metadata, while the accesses to the shared and shared+
Figure 4.13: Memory durability (as petabyte write) versus the average error per page for different error correction schemes. Using block cooperation increases the number of correctable errors per page and increases metadata utilization for both ECP and Aegis.
blocks are performed with no extra latency and further memory access. In Aegis, when using data layout reorganization, every read from the cooperative blocks requires reading data and metadata from other blocks. For writing to the cooperative blocks, reading data from the other blocks is required, involving further metadata updates. Therefore, error recovery can take more cycles in Aegis as compare to ECP with block cooperation.

4.5 Summary of Block Cooperation

Once an error recovery scheme fails to recover from faults in a data block, the entire physical page associated with that block is disabled and becomes unavailable to the physical address space. To reduce page waste caused by early block failures, we proposed block cooperation technique, where other blocks help failed blocks to remain alive and extend the page’s lifetime.

We combined the proposed technique with two state-of-the-art error recovery schemes, ECP and Aegis. The goal was to increase memory lifetimes, with only minor modifications to the underlying error correction mechanism. We evaluated our propose modifications to ECP (metadata sharing) and Aegis (data layout reorganization). We used both Monte-Carlo and trace-driven simulation to evaluate our approach. We also presented a workload characterization scheme that focuses on memory lifetimes, providing insight on how workload behavior is tied to the success of different error recovery schemes.

Employing block cooperation on top of Error Correction Pointer (ECP) and Aegis increased memory lifetimes by 37%, and 14% on average, respectively. Lifetimes can be increased further by 60% (68%) by exploiting metadata sharing, or through data layout reorganization 13% (30%), considering CoV of 0.2 (0.3), respectively.
Chapter 5

Cost-Effective Write Disturbance Mitigation Techniques

In this chapter we propose two general-purpose, and cost-effective, solutions to mitigate write disturbance (WD). The first technique considers Data Modification with Partitioning (DMPart), a technique that divides a data block into smaller chunks, and encodes data using predefined patterns to reduce the number of consecutive zeros. This can significantly reduce the probability of a write disturbance. The second technique, which is called Selective Write to the Exposed cells (SWEX), overwrites vulnerable cells if the number of vulnerable cells is below a set threshold. Our results find a 32% and 49% reduction in the average number of writes using DMPart, and SWEX techniques, respectively. We show that the impact of our techniques in terms of energy consumption of the memory system, and the memory lifetime, is insignificant or even improved.

The remainder of this chapter is organized as follows. Section 5.1 provides our write disturbance fault model. Section 5.2 presents previous schemes to mitigate write disturbance. Section 5.3 presents our proposed schemes. Section 5.4 discuss our experimental results on average number of writes, energy and lifetime. Finally, Section 5.5 concludes the chapter.

5.1 Write Disturbance Fault Model

Figure 5.1 illustrates an example of a disturbed write. Writing zero bit values potentially can disturb cells in their vicinity, provided that the neighboring cells contain an idle zero. In Figure 5.1, we name the former disturbing cells and the latter as exposed cells. Note that if a Data Comparison Write (DCW) strategy is employed in PCM, every ‘1’ → ‘0’ transition in a cell is
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Considered to be disturbing cells (the two dashed blue cells). There are also three exposed cells that, after the write operation, one of them is disturbed and changed to ‘1’ (cell #7 in Figure 5.1).

We consider the WD model presented by Jiang et al. [42], which includes a PCM scaling model, a PCM cell thermal model, and a PCM thermal disturbance model. Similar to Jiang et al. [42] and Wang et al. [88], we focus on 20nm technology. We assume that the minimal distance between the PCM cells (the pitch size) is $2F^1$, the thermal disturbance error rate for SLC PCM cells is 9.9% [42].

5.2 Previous techniques

Next, we consider potential solutions to tackle the write disturbance problem in PCM. These techniques are used to mitigate disturbance along a word-line. First, we briefly describe each solution, and then consider tradeoffs associated with each technique.

i) Verify and Correct (VnC) [19]. When writes are performed in PCM, heating occurs, which is a non-deterministic operation. Using VnC, a verification read is performed after each write to ensure the cell’s resistance remains in the desired range. VnC involves no extra support to provide for WD mitigation, and instead allows the PCM chip to correct disturbed cells, one by one, or in multiple verification/correction operations. The main drawback of this class of schemes is a dramatic performance impact, especially when the probability of a disturbance is high.

---

$^1F$ is feature size.
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Figure 5.2: Writing to all the exposed bits to eliminate any write disturbance results in a domino write to all the neighboring idle ‘0’ cells (green dashed square).

ii) Write to all the EXposed cells (WEX) \([41]\). This scheme relies on a write comparison \([95]\) algorithm that compares the new data block with the old one to find the exposed cells. To eliminate WD, this scheme RESETs the exposed bits to prevent heat dissemination from altering the value of the exposed bits. This approach is able to completely remove any disturbance, but the negative side-effects outweigh any benefits. First, writing a zero by issuing a RESET pulse dissipates 25% to 150% more power (as compared to a SET pulse). Second, the limits on current delivery of the charge pumps in the PCM peripheral determines the number of parallel cell writes. For example, in a 20 nm PCM prototype, 128 cells can be written in parallel \([20]\). As we increase the number of written cells, we increase the number of shots (i.e., extra write cycles) required to write the whole data block (i.e., 512 bits). Third, a small modification in a data block can result in a domino effect, generating multiple unnecessary cell writes to the data block. This accelerates cell wear-out. Figure 5.2 illustrates this phenomenon, resulting in in many extra bit writes. Given a sequence of zero bits in the data block (which is common in many applications), all of the bits need to be RESET due to the RESET of the exposed bit (cell #6), in order to avoid WD.

iii) Data encoding based INsulation technique (DIN) \([41]\). In contrast to the WEX scheme, which aims to remove WD completely, DIN tries to reduce the probability of vulnerable data patterns using data encoding. In order to avoid extra storage overhead, DIN relies on data block-based compression to make room for encoded data. DIN encodes every 3-4 bits, significantly reducing the chances of encountering two consecutive zeros in a block. Because this technique only lowers the chances of a WD, it adds a 20-bit BCH code to correct up to two errors in the encoded data.

The main drawback of DIN is that it is not a general solution. DIN’s efficiency is reduced
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

if a data block is not compressible. Uncompressed blocks are not encoded, and so VnC techniques are employed to ensure the correctness on write operations for such blocks. Figure 5.3 shows a group of SPEC2006 benchmarks, where DIN is able to encode portions of the data blocks written to main memory. Due to poor compressibility in some of benchmarks, such as leslie3d, hmmer, and milc, only 20-30% of data blocks are compressible by DIN. Frequent Pattern Compression (FPC) [4] is used for compressing data blocks in DIN.

Similar to the focus of this work, all the schemes discussed above mitigate write disturbance along the memory word-lines. Writes may disturb adjacent lines in a PCM if the distance between bit-lines is very small. SD-PCM [88] addresses WD along bit-lines by proposing Lazy-Correction. This technique uses DIN to reduce disturbance along word-lines, and exploits unused Error Correction Pointer (ECP) [72] entries to correct disturbed cells in adjacent lines whenever possible. LazyCorrection requires a low density ECP chip to avoid WD problems in the correction chip. Moreover, to further reduce performance loss, two pre-reads, to adjacent lines associated with the written line, are scheduled in idle periods. These pre-reads fetch the contents of the adjacent lines into a buffer, which triples the size of the write buffer. Because we only focus on WD along word-lines, comparison with SD-PCM [88] is not considered in this work.

5.3 Proposed Schemes

In this section, we propose two techniques to mitigate WD in high density PCMs. A pragmatic approach to tackle WD necessitates generality, simplicity, and limited side-effects. Our definition of generality means that the technique needs to function effectively in general cases,
without any preconditions. Simplicity considers the overhead of the implementation. Side effects include other metrics in terms of memory architecture or overall system impact (e.g., lifetime, energy, performance, etc.). Our goal is to provide schemes which can address generality, simplicity and side-effects together. The first scheme tries to reduce the chance of WD by modifying the data pattern of the written data. The second scheme, selectively writes to the exposed cells, but considers the cost of the extra bit write(s), and the probability of a disturbance.

5.3.1 Data Modification with Partitioning (DMPart)

The WD problem in PCM is dependent on the data pattern of data written to main memory. DMPart proactively tries to reduce the number of consecutive zeros in data using different pattern manipulators. A pattern manipulator encodes data before writing it to main memory. The pattern used is stored as metadata for decoding purposes.

In Figure 5.4, we use a simple example to show how DMPart is used to mitigate WD. A data block is divided into \( n \) contiguous and equal-sized partitions. Without loss of generality, we assume the data block is only 4B, partitioned into four 1B partitions. Each partition is encoded separately with a dedicated pattern manipulator. This scheme considers four manipulators, which consist of the following 2-bit patterns: ‘00’, ‘01’, ‘10’, ‘11’. The number of repetitions of each pattern in each partition is calculated and the pattern with the minimum frequency is selected as the manipulator pattern (Figure 5.4a). Note that finding the manipulator pattern is performed only on write accesses, which are not in the critical path of execution. Figure 5.4c shows the manipulator selection logic for partition #1. The pattern decoder records each occurrence of the different patterns in the data partition. Then, the output is fed into the minority circuit logic to select the pattern with minimum frequency. Generating the manipulator pattern is performed separately for each partition.

On read accesses, the data needs to go through the encoder/decoder logic (Figure 5.4b), which is basically a single level of 2-input XOR gates. On write accesses, the data is simply encoded by XOR-ing the data with the manipulator pattern (Figure 5.4d). Similarly, on read accesses, the data is decoded by the same logic.

Selecting a pattern with minimum frequency results in minimizing the number of zero sequences in the encoded data. In our example, the ‘10’ pattern, has the minimum frequency, and is selected as the manipulator pattern. Therefore, instead of writing the original data ‘01110000’, we would write ‘11011010’ to main memory, along with two-bit pattern as the metadata necessary for reconstructing the original data. Note that counting the frequency of patterns is performed two
Figure 5.4: (a) Manipulator pattern selection, as well as data encoding for different partitions, which is performed on write accesses. Data is encoded before being written to the main memory to reduce the chance of consecutive zeros. (b) Decoding data per each partition when it is fetched into the chip. (c) Manipulator selection logic. (d) Encoder/Decoder logic.

bits at a time, hence, selecting less frequent patterns does not guarantee the minimum number of zero sequences in the encoded data. However, accepting this limitation reduces the number of zero sequences effectively, while limiting complexity and overhead in the design. The storage overhead can be tuned by selecting the proper partition size (e.g., an 8B partition leads to 16 bits of storage for a 64B data block, or 3.125% overhead).

Our DMPart scheme stores manipulator patterns in the ECC chip. The Error Correction Pointer (ECP) [72] and Aegis [28] schemes are among the error correcting codes that have been proposed for PCM to recover permanent cells faults. The former allocates ECP entries (each consists
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Figure 5.5 shows the ratio of the average number of exposed bits when VnC is used, as compared with DMPart. A higher value (above the red dotted line) indicates that DMPart is more effective in reducing the number of vulnerable data patterns, therefore reduces the number of write shots. However, there is no guarantee that DMPart reduces the average number of exposed bits. For instance, for sjeng and milc, where DMPart is below the red line, the WD problem dominates.

5.3.2 Selective Writes to EXposed cells (SWEX)

The WEX scheme eliminates WD by writing zeroes in all exposed bits. However, the side-effects associated with WEX make this technique impractical for mitigating WD. In PCM, the number of writes, from a device’s point of view, can be more than the actual writes that are issued by the application. Extra write shots in a PCM device can be for a number of reasons.

i) Device limitations: PCM write operations require a considerable amount of power, which is provided by the charge pump circuit. Due to the large current generated during a write, simultaneous bit writes are limited to avoid undesirable current spikes and power noise. For instance, in a 20 nm PCM prototype [20], a 128b parallel write operation is supported (a 256b parallel write is
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Figure 5.6: Relationship among error rate, number of exposed bits and the write failure probability. When the number of exposed cells is high, it is not cost-effective to lower the write failure probability by writing to them.

possible, if provided with external power). Using a comparison write \cite{95}, together with a Flip-N-Write \cite{18} scheme, we can reduce the number of bit writes, and reduce the performance overhead incurred due to parallel bit writes in PCM.

ii) Write Errors: Write operations may not be successful in the first shot, for a variety of reasons. Process variations increase the uncertainty of bit write operation. Moreover, a WD can prevent us from performing a successful write in one shot. Applications show different sensitivities to WD, since vulnerability is data-pattern sensitive.

Figure 5.6 presents the relationship between the: i) the error rate (the x axis), ii) the number of exposed cells (the y axis), and iii) the write failure probability (the contour levels shown using different colors). The error rates for SLC/MLC modeled assuming 20 nm PCM, are shown using vertical dashed lines. Figure 5.6 shows that when the number of exposed bits is high, it is neither practical, nor cost-effective, to reduce the number of exposed bits. For instance, there is still a greater than 90% chance of a write error, when the number of exposed bits is reduced from 100 to
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Algorithm 1: SWEX Scheme

Input : bit[512] new_data, bit[512] old_data, bit[512] exposed_bits, integer \( \eta \)

Output : bit[512] write_mask

1 exposed_bits ← WEX.exposed_bits(new_data, old_data)

/* find exposed bits with domino effect */
2 if (\# of exposed bits \( \leq \) \( \eta \)) then
3 write_mask= exposed_bits
4 else
5 for <each bit \( i \) in exposed_bits> do
6 write_mask[i]= exposed_bits[i] AND (exposed_bits[i-1] NOR exposed_bits[i+1])
7 end
8 end

50. The distance between contour levels becomes smaller when the number of exposed bits becomes small. This observation indicates that when the number of exposed bits is smaller, writing to cells in order to eliminate WD is more effective, and comes at a lower cost.

Algorithm 1 shows results for our SWEX scheme. The output of SWEX is a write_mask, which identifies which of the exposed bits should be written, along with the data to reduce WD. Ones in the write_mask enable writes to the exposed cells. Our SWEX scheme is based on the WEX scheme. Therefore, all of the exposed bits, including cells that could be prone to the domino effect, should be identified first (line 1). Then, if the number of exposed bits (the number of ones in exposed_bits) is less than or equal to a predefined threshold (\( \eta \)), all of the exposed bits are marked to be written (line 3). Otherwise, only the exposed bits are selected for writing (line 5 and 6), avoiding the domino effect. Note that both adjacent cells of an exposed bit should be zero in exposed_bits in order to select that cell for extra zero over-writing. The \( \eta \) parameter in SWEX helps to trade-off write errors versus cost (i.e., performance, energy, and life-time).
Table 5.1: The system specifications of our simulated system.

<table>
<thead>
<tr>
<th>Processor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>X86</td>
</tr>
<tr>
<td>CMP</td>
<td>4-core, out-of-order, 2.66GHz</td>
</tr>
<tr>
<td>Interconnect Network</td>
<td>Bus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>On-chip Cache Hierarchy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>256KB/64B/8 way, private</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>8MB/64B/16 way, shared</td>
</tr>
<tr>
<td>Store Policy</td>
<td>Write-Back</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PCM Main memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Capacity</td>
<td>4GB (4KB pages, and 64B rows)</td>
</tr>
<tr>
<td>Configuration</td>
<td>4 memory controller</td>
</tr>
<tr>
<td></td>
<td>1 rank/DIMM, 8 devices/rank,</td>
</tr>
<tr>
<td></td>
<td>1 chip for ECC, ECP [72] error correction</td>
</tr>
<tr>
<td>Endurance</td>
<td>Mean: $10^8$ writes, Variance: 0.25</td>
</tr>
</tbody>
</table>

5.4 Evaluation and Results

5.4.1 Experimental setup

We use SniperSim [13] to execute workloads and generate memory traces of applications from the SPEC2006 benchmark suite [77]. Table [5.1] describes the parameters used in the modeled system. Our baseline is a 4-core out-of-order multiprocessor which includes three levels of on-chip cache in the memory hierarchy. The benchmarks are executed in rate mode [60], where all the four cores execute the same benchmark.

Write requests are captured at the PCM memory, only on an eviction from the last level cache (i.e., L3 in our simulations). The granularity of read and write requests are 64 bytes (i.e., one cache line), and are serviced by one of the PCM banks. The first 2 billion instructions for each benchmark are used to warm up on-chip caches, and then 10 million references to main memory, along with 64 bytes of data for each access, are captured. Note that 10 million references corresponds to several billions of instruction execution for each benchmark. Write accesses are more
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Figure 5.7: Benchmark sensitivity to write disturbance. We evaluate the effectiveness of our proposed schemes, using benchmarks which have moderate to high sensitivity.

important, as the proposed schemes are designed to mitigate WD, and are performed before writing to the main memory.

WD is highly dependent on the pattern of the data written to the PCM memory. We evaluate benchmarks in the SPEC2006 benchmark suite using VnC to gauge sensitivity, in terms of WD. As shown in Figure 5.7, some of the benchmarks only experience a minor disturbance, resulting in less than a 20% increase in the number of writes. We classify them as low sensitivity benchmarks, and consider the rest as moderate to high sensitivity benchmarks. We report results for only the latter class of benchmarks, which exhibit more than a 20%-120% write increase. We also assume VnC as our baseline scheme, and normalize results with respect to VnC results.

5.4.2 Results

5.4.2.1 Average Number of Writes

The average number of writes, or the expected value of writes, is the main metric which shows the effectiveness each scheme. The expected value of writes indicates how many write accesses (on average) are necessary to service one write request in a workload. Figure 5.8 shows this metric for various benchmarks when different schemes are employed. VnC is our baseline. DIN and DMPart modify the data pattern to reduce or remove vulnerable patterns. WEX tries to eliminate disturbance by writing to all the exposed bits, and finally SWEX selectively writes to the exposed bits based on η parameter (i.e., η is set to 32 in our evaluations).

In most benchmarks, DIN and DMPart outperform VnC. However encoding written data to PCM does not guarantee better results, as found in a few of benchmarks such as milc, perlbench or calculix. For these workloads, the rate of WD increases after employing DIN. SWEX outperforms DMPart and DIN, and guarantees fewer writes than VnC. Overall, the geometric
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

Figure 5.8: Expected value of writes. SWEX shows more robust performance, as compared to other schemes.

mean of the number of writes for the studied benchmarks is 1.53, 1.41, 1.38 and 1.24, when using VnC, DIN, DMPart and SWEX, respectively.

5.4.2.2 Energy Consumption

Energy consumption is another important metric for a memory system. Employing comparison writes in PCM reduces the dynamic power for write accesses drastically. DIN and DMPart, which modify the data written to main memory, can potentially increase the number of bits written, which increases energy consumption. On the other hand, WEX over-writes zeroes in many cells to remove WD, which consumes a considerable amount of energy.

We adopt the energy model and the power parameters used by Zhou et al. [95] for PCM to evaluate the energy costs. To find the impact of each scheme in terms of energy consumption, we classify PCM operations as either read, write or verify operations. We use Equation (5.1) to model energy consumption for PCM.

\[
E_{PCM} = N_{read}E_{read} + N_{write}E_{write} + N_{verify}E_{verify}
\]  

(5.1)

For each workload, \(N_{read}\), \(N_{write}\), and \(N_{verify}\) are the number of reads, writes, and verify operations, respectively. \(E_{read}\) are generated to read a 64B block from PCM, which is approximately equal to 1.075 nJ. The verification process involves another read operation (i.e., verify read), which is performed after each write to make sure the data has been written correctly. Due to the comparison
CHAPTER 5. COST-EFFECTIVE WRITE DISTURBANCE MITIGATION TECHNIQUES

write used in PCM, the write energy can be modeled as Equation (5.2) [95].

\[ E_{\text{write}} = E_{\text{fixed}} + E_{\text{read}} + E_{\text{bitChange}} \]  

(5.2)

where \( E_{\text{read}} \) is the energy for the read operation required to read out the old data and place it in a memory buffer, \( E_{\text{fixed}} \) is the fixed amount of energy consumed for write decoding, row selection, data comparison (using XNOR gates), etc. \( E_{\text{fixed}} \) is 4.1 nJ per access. Finally, \( E_{\text{bitChange}} \) models the actual cell writes for a block of data, which is included in Equation (5.3) [95].

\[ E_{\text{bitChange}} = N_{0\rightarrow1}E_{0\rightarrow1} + N_{1\rightarrow0}E_{1\rightarrow0} \]  

(5.3)

where \( N_{0\rightarrow1} \) and \( N_{1\rightarrow0} \) is the number of bit flips from 0 to 1 and vice-versa, respectively. \( E_{0\rightarrow1} \) and \( E_{1\rightarrow0} \) are the amounts of energy consumed when writing a ‘1’ or ‘0’, which are 0.0268 nJ and 0.013733 nJ, respectively. Note that in the WEX and SWEX schemes, since some of ‘0’ bits are overwritten, we take this into account as extra ‘0’ writes.

Using the energy equation Equation (5.1), the breakdown of energy consumption for different benchmarks is shown in Figure 5.9. All of the results are normalized versus the VnC scheme, and the changes in each energy component is shown. The red dotted line shows the baseline (i.e., VnC). Read energy stems from the read accesses in the workload, and is constant, irrespective of which scheme is used for WD mitigation. Verification energy directly depends on the number of writes accesses in each benchmark, and the number of extra write shots imposed by WD. The write energy can be changed by DIN and DMPart, where encoding data is used for disturbance mitigation. Encoding data can increase the number of bit flips, thereby increasing write energy. For instance, a frequent pattern compression algorithm FPC [4] is used for compressing each 4-byte chunk of data in a 64B block of cache line in the DIN scheme. If one of the input chunks for FPC is not compressible, all the encoded bits in the rest of the cache block will be changed, consequently

Figure 5.9: Breakdown of the energy consumption, normalized to the VnC scheme.
Figure 5.10: Bit flips across the different write disturbance mitigation techniques. Each result is normalized to VnC. DMPart uses 8B partitions, and $\eta$ is set to 32 in SWEX.

resulting in more bit flips and added energy consumption. The same case can happen for DMPart, where small modifications in data values result in significant changes within a encoded partition.

WEX reduces the verification energy, as compared to VnC. However, the extra bit writes, multiplied by any domino effects, increase the energy consumption for writes drastically. For instance, although the verification energy is less than that of VnC in the astar and gobmk workloads, the write energy becomes significantly higher, which results in 39% and 50% higher energy consumption, respectively. On the other hand, SWEX reduces the verification energy by reducing the number of write shots, incurring a smaller increase in write energy. The total energy consumption for SWEX is less than that of VnC in many cases. On average, WEX, DIN, and DMPart consume 15%, 21%, and 11% more energy, as compared to VnC, while SWEX reduces energy consumption by 4%.

5.4.2.3 Bit Flips and Lifetime

Each PCM cell endures only a limited number of writes [72, 84]. After passing the endurance limit, PCM loses the ability to store new data. A technique which tries to mitigate WD should not have a considerable impact on wear-out.

Figure 5.10 shows the normalized bit flips, as compared to VnC. WEX causes an excessive number of bit writes as it doubles or triples the cell writes in a few of the benchmarks. Note that overwriting ‘0’ in WEX has a similar impact of ‘1’ to ‘0’ bit flips. By encoding data, DIN and DMPart lower the effectiveness of the comparison write and increase bit flips, especially given that the number of cell writes increases by more than 4X in libquantum when DIN is employed. Although we see
Figure 5.11: Page lifetimes in writes-per-page, when different schemes are used for write disturbance mitigation.

a 20% increase in cell writes for libquantum benchmark with SWEX, the geometric mean of cell writes increases by less than 6%. The mean values for WEX, DIN, and DMPart are 80%, 44%, and 49%, respectively.

To model the lifetime of PCM, we employ the same methodology used in Schecter et al. [72], and Tavana et al. [84]. The lifetime is assumed to follow a normal distribution, with a mean $\mu$ of $10^8$, and with a coefficient variation of 0.25 in terms of the mean cell lifetime. Similar to previous work [28,72,84], we assume perfect wear leveling across the memory blocks, which is in-line with wear leveling techniques previously proposed for PCMs [65]. We assume ECP [72] is used for error recovery, and whenever ECP cannot correct the faults in a memory block, the whole page containing the memory block is removed from the accessible memory address. Figure 5.11 plots the page lifetime when different schemes are used for WD mitigation. The reduction in lifetime of SWEX is only 5% as compared to VnC. PCM lifetimes are reduced by 44%, 31%, and 33% when WEX, DIN, and DMPart are used, respectively.

5.5 Summary of Write Distrubance Solutions

Write disturbance is the main obstacle for PCM to become the technology of choice for future high capacity memory systems. Allocating large inter-cell space to eliminate write disturbance results in drastic capacity reduction in PCM chips. In this chapter, we proposed two cost-effective
techniques to reduce the probability of write disturbance. The first technique (DMPart) encodes the data written to PCM in order to eliminate consecutive zeros and reduce the chance of having vulnerable data patterns. The second technique (SWEX), detects exposed cells and overwrites them if the number of cells is below a selected threshold. Otherwise, SWEX only overwrites cells which do not cause domino writes.

The proposed techniques reduce the number of extra writes caused by write disturbance. Our experimental results showed that DMPart and SWEX reduce the average number of writes up to 32% and 49%, respectively, while inuring minimal impact on the lifetime and energy consumption of PCM.
Chapter 6

Durable, Secure and Energy-efficient Non-Volatile Memory

Although non-volatility is a desirable property in terms of power savings, it also introduces new security vulnerabilities to the system. Sensitive information can persist in non-volatile main memory, even when the system is turned off. Therefore, data at-rest in NVMs should be protected against physical memory readout. Memory encryption can be used to protect data confidentiality, and counter-mode encryption is a commonly used scheme since it allows data to be fetched in parallel to cryptographic pad generation \[76\]. The pad is simply XOR’ed with the fetched data to perform decryption. The pad value is unique, and should only be used once for each memory block. Therefore, a counter or sequence number is assigned to each memory line to secure the data.

Memory encryption has been successfully integrated in recent commercial microprocessors, including the Intel Skylake \[31\] and the AMD Zen \[45\]. However, typical memory encryption engines are not adequate for non-volatile memories, where every bit write is expensive. A key design principle in any encryption algorithm is property of strong diffusion \[78,80\], i.e., when a single bit change occurs in the plaintext, multiple/many bits will flip in the ciphertext (statistically, half of the bits). Even with the DCW technique, such pseudo-randomizing bits in each memory writeback will result in excessive overwrites and memory lifetimes will be impacted. Moreover, this will also lead to higher energy consumption, which may outweigh any benefits that are achieved from using non-volatile PCM.

To better understand the impact of encryption on data comparison write (DCW) in PCMs, we consider three writes to main memory in Figure 6.1 taken from the astar benchmark. When
Figure 6.1: Consecutive writes of an encrypted (a), and unencrypted (b) memory block for the astar benchmark. Encryption increases the number of bit flips.

Memory is unencrypted, only 4 bytes in a 64 bytes cache line are modified, with 11 bit flips in each writeback. When using encryption, the same writeback of the cache line can cause 252 and 261 bit flips (roughly half of 512 bits) in our example. As PCM can only endure a limited number of writes \(10^5-10^9\) \[50\], the extra bit writes due to encryption reduces memory lifetime significantly.

Figure 6.2 shows the lifetime reduction and energy increase for an encrypted PCM as compared to using unencrypted PCM for four benchmarks. For the libquantum benchmark, the lifetime and energy consumption of PCM are significantly impacted (11.2x and 58%, respectively). Even in the best case, PCM lifetime is reduced by 2.6x and energy increased by 24%, as observed in the omnetpp and hmmer benchmarks, respectively.

The goal of our work is to propose a mechanism to bridge the lifetime gap between unencrypted and encrypted memory systems. To this end, we propose Nacre, a novel design for secure NVMs to significantly reduce the number of bit writes needed, and consequently improve the lifetime and energy consumption of NVMs. Nacre breaks down a cache line into multiple segments, and keeps track of different versions of the changes to each segment. By utilizing short segment counters, combined with a small amount of metadata to record the history of data modifications, we can significantly reduce the number of bit writes to modified segments.

We evaluate Nacre on PCM, and believe the techniques presented in this chapter, are also

\[^1\]Our methodology for lifetime and energy analysis are presented in Section 6.2
Figure 6.2: Lifetime reduction and energy increase for encrypted PCM memory vs. unencrypted PCM memory. The Y-axis on the left shows the lifetime reduction (solid bars), and Y-axis on the right shows the energy increase (dashed bars), normalized to the unencrypted memory.

effective for improving lifetimes in other NVM technologies (including RRAM and STT RAM [25]). We show that on average, Nacre reduces the number of bit flips, improving memory lifetime, by 53% as compared to state-of-the-art schemes. In addition, Nacre only increases energy consumption by 6% versus an unencrypted memory system.

The rest of this chapter is organized as follows. Section 6.1 provides our proposed write-efficient security architecture for NVMs. First, we start with introducing our target architecture and attack model, then describe state-of-the-art architecture for securing main memory. In Section 6.1.2, we provide the details of Nacre, our proposed security architecture. Simulation methodology and the results are provided in Section 6.2. Finally, Section 6.3 concludes the chapter.

6.1 Proposed Security Architecture for NVMs

6.1.1 Background and Assumptions

6.1.1.1 Attack Model

Figure 6.3a shows our target system, with a multi-core processor connected to a PCM-based main memory through the system bus. We assume the attacker has physical access to the system and is able to tamper with main memory and detect data values streaming out to memory. The components within the chip are running in a secure domain, and the off-chip components, including the memory bus and main memory are not secured. Nacre is designed to secure the system against bus
snooping or memory readout by encrypting the data transferred out of the secure domain. Moreover, Nacre is able to protect a system against known-plaintext attacks (observing both the plaintext and the ciphertext), and dictionary-based attacks (predicting the original values based on the frequency of encrypted values).

6.1.1.2 PCM-based Main Memory

Figure 6.3b shows a PCM-based Dual In-line Memory Module (DIMM). A PCM rank comprises nine chips, where the ninth chip stores an Error Correcting Code (ECC) for the other eight data chips. The data associated with a 64 byte cache line is interleaved across all of the chips in the DIMM, which takes eight burst accesses to read/write from/to main memory. For error correction, we assume a specific scheme such as ECP [72] is used to store the error correction metadata in the ECC chip.

Since write operations are costly in PCM and result in excessive energy consumption and reduced lifetimes, Data Comparison Writes (DCWs) [95] are used to limit bit writes to only the differences in a cache line (versus writing every bit in the cache line, irrespective of its value). More
CHAPTER 6. DURABLE, SECURE AND ENERGY-EFFICIENT NON-VOLATILE MEMORY

specifically, a read-modify-write circuit [95] is provided on PCM chips to first read the old data, find the differences with the new data, and only write to the modified bits in the memory block.

6.1.1.3 Memory Encryption

In secure memory systems, whenever data is written back to main memory, the Crypto Engine (Figure 6.3a) is responsible for encrypting data using a standard encryption algorithm such as AES (Advanced Encryption Standard). Whenever the data is read from memory, the system decrypts data before moving it to the cache hierarchy and the processing cores. There are different implementations to achieve memory encryption. Counter-mode encryption (CTR-mode) is employed in state-of-the-art memory encryption schemes [7, 31, 45, 48, 76, 92].

CTR-mode encryption has two interesting characteristics which we exploit to design a high performance, secure and durable, non-volatile memory. First, CTR-mode encryption transforms a block cipher into a stream cipher, where the plaintext data does not need to be a multiple of the cipher block size. CTR-mode encryption works just as well for one bit of data, as it does for an entire cache line of data [78]. We take advantage of this property to reduce the number of bit modifications needed to encrypt data, which in turn, reduces the impact of encryption on memory lifetime.

Second, with CTR-mode encryption, instead of applying the encryption algorithm directly to the data, CTR-mode encryption generates a One-Time Pad (OTP) based on the initialization vector (IV) and key, which can be run in parallel with data retrieval, as shown in Figure 6.3c. OTP is simply XORed with the data for encryption/decryption. Since CTR-mode encryption hides memory latency, it is a very good fit for secure memory in high performance systems [76, 90].

Employing CTR-mode encryption comes with some requirements in order to provide a secure system. The generated pad should be only used once, and also should guarantee spatial and temporal exclusivity [7, 76, 80, 90, 92]. Spatial exclusivity means that pads should be unique across different lines in memory. Temporal exclusivity means that pads should be unique across different values of a specific line over time. The former is achieved by using the address of each memory line as part of the IV, and the latter is achieved by assigning a counter to each memory line in the cache (64 byte block). Figure 6.3c shows IV and how CTR-mode encryption works to encrypt/decrypt data. Note that the IV is independent of contents of a data block, therefore generating a pad can be initiated in parallel with the data fetch. Whenever a memory line is written back to the main memory, the associated counter is incremented to change the IV. In the case of a line counter overflow, the page counter is incremented, and all the lines in that page are re-encrypted.

84
Counters are stored in the main memory, however the most frequently used counters are also cached on chip near the cryptography engine to help hide decryption latency (Figure 6.3c). Note that the IV is not secret and revealing the counter information is not a security concern, therefore the counters can be stored in an unencrypted form. Nonetheless, counters integrity should be guaranteed in order to prevent attackers from tampering with counter values, and repeating a counter value for a cache line (i.e., the pad reuse problem). To this end, counters can be protected with a Bonsai Merkle Trees [70] authentication scheme, which introduces very little overhead.

6.1.2 Nacre Design

Next we describe Nacre, a secure NVM design which can significantly reduce the number of unnecessary bit writes, and consequently improve durability and energy consumption of NVMs. Nacre breaks down a cache line into a number of segments and tracks modifications to each of these segments. Utilizing small segment counters, along with a small amount of metadata to track the history of data updates, we can better manage the number of bit writes to modified segments.

Note that in Nacre, multiple OTPs may need to be generated, using multiple segment counters and different AES blocks. The stored metadata tracks which segments should be XOR’ed with which OTP, in order to encrypt or decrypt the cache line. To better illustrate the utility of Nacre and explain the role of the metadata, we present a simple example in Figure 6.4 walking through the steps followed at each writeback event.

We introduce segment counters (SCs) to generate different OTPs for the current modified segments and the previous versions of segment modifications in a cache line. In this example, we assume three 2-bit SCs are used, with each segment 8 bytes in length (8 segments in a 64-byte cache line). Three SCs are used as part of the IV in order to generate three different OTPs in parallel, each using different encryption/decryption blocks. We simply need one bit for each segment in the last level cache (LLC) to identify which segment was modified during the lifetime of the data in the LLC. We refer to this set of bits as the Segment Modification Indicator (SMI) in our design. For each segment counter (SC0-SC2), one OTP mask equal in size to the size of the SMI is required to manage the multiple writebacks. The OTP mask records which segments in a cache line are required to be XOR’ed with its corresponding OTP value. The three masks should be exclusive to each other, and satisfy \( \text{Mask}_0 \oplus \text{Mask}_1 \oplus \text{Mask}_2 = 11111111 \). Therefore, based on the OTP masks, an encrypted/decrypted cache line can be generated by XORing different segments of the OTPs with the corresponding segments of data.
Figure 6.4: A simple example which shows Nacre’s function at writebacks events step by step. SMI shows modified segments at each writeback. At each step one of the segment counters (SCs) is selected and the masks metadata are updated accordingly.
We start in Step 0, initializing the state of the SCs and their corresponding OTP masks. Initially, all the SCs are set to zero and OTP₀ (the OTP generated by SC₀) is used for all segment encryptions. In step 1, the first writeback occurs and the SMI indicates that only segments 0 and 5 were modified. SC₁ is selected to record the first version of line modifications, and the associated OTP mask is updated accordingly. Bits 0 and 5 in mask₁ are set, and the corresponding bits in OTP mask₀ are reset.² The selected segment counter (SC₁) should be incremented. Note that based on the current state of the OTP masks, segments 1,2,3,4,6, and 7 of OTP₀ are used for encryption of the unmodified segments of the cache line, and segments 0 and 5 of the OTP₁ are used for encryption of the modified segments. Only the modified segments (0 and 5) are re-encrypted, and thus introduce bit flips in the memory cells.

In step 2, the second writeback occurs. The cache line segment counter, SC₂, is selected for the newly changed segments. Similar to step 1, the OTP masks need to be updated and the SC₂ is incremented as well. Note that segments 0, 1, and 2 are encrypted using the OTP₂ mask generated by SC₂, SC₁ only encrypts segment 5, and the rest of the segments are encrypted using the OTP₀ mask generated by SC₀.

In step 3, the third writeback occurs. There are no free SCs to assign to the most recent cache line updates. Therefore, this version needs to be merged with one of the previous versions. Our policy for merging is to select the counter with minimum number of ‘1’s in the corresponding OTP mask. Note that ‘1’ bits in the selected OTP mask track the modified segments from previous versions of memory writebacks. By merging versions, these segments, along with the modified segments of current version, should be re-encrypted. The heuristic behind our selection policy is to minimize the extra encryptions caused by merging masks. In our example, SC₁ includes a single ‘1’ in its mask and is selected for merging with the current version. However, due to an overlap between the modified segments in the current version (segments 4 and 5) with the segment being tracked by SC₁ (segment 4), there is no extra re-encryption after merging versions.

The update of masks is handled as follows. Assuming N is the set of SCs used in a Nacre design, updating the mask associated with a selected SC is

\[ mask_k \leftarrow mask_k \lor SMI \]  

(6.1)

where \( k \) is the selected SC. Updating other masks can be expressed as

\[ mask_i \leftarrow mask_i \land \lnot SMI \]  

(6.2)

²The bit modifications in masks are marked with bold and larger font.
where \( i \in N \), and \( i \neq k \). Similarly, the modification in the fourth writeback in step 4 needs to be merged with one of the previous versions. Once again, \( SC_1 \) is selected, since it has fewer ‘1’s. Next, masks are updated and \( SC_1 \) is incremented to ‘11’ accordingly. In step 5, \( SC_0 \) is selected. The masks are updated and \( SC_0 \) is incremented. Finally, in 6, \( SC_1 \) is selected once again. Incrementing \( SC_1 \) results in an overflow, therefore all the SCs are reset, the line counter is incremented, and all the segments are re-encrypted subsequently (step 7). Note that if a line counter overflows, the associated page counter should be incremented, and the entire page should be re-encrypted accordingly.

The mask metadata needs to be saved beside the data in the memory. However, whenever an LLC miss occurs, we need to fetch the mask data in order to decide which OTPs, and which OTP segments, are to be used for decryption of the entire cache line. We can use an ECC chip in the PCM DIMM (Figure 6.3b) to hold this metadata, since the ECC bits in the PCM are not utilized unless data blocks experience wear-out faults. For instance, error correcting pointers (ECP) [72], which have been proposed for error recovery in PCM, incur a 10-bit overhead to recover one bit error, but are able to correct 6 errors per line. However, only a few memory lines utilize all the ECP entries [84], which motivates us to re-purpose the ECC chip to store mask metadata. Note that if the number of errors for one line is high, we simply discard the saved mask metadata, and instead full-line encryption is applied. Note that masks data does not need to be stored encrypted, since the security of counter mode encryption does not depend on that.

In our example, we require 24 bits to store mask metadata. In general, for \( N \) segment counters and \( M \) segments, \( M \times N \) bits are required to store mask metadata in Figure 6.4. However, in our storage-efficient implementation of Nacre, a counter value is assigned to each segment to keep track of which counter is used for that segment. Our storage-efficient Nacre implementation only requires \( M \times \lceil \log_2 N \rceil \) bits to save the masks.

### 6.2 Evaluation and Results

To begin our evaluation of Nacre, we first look for the best segment size and the right number of counters to use. Then, we compare Nacre with competing designs, utilizing a range of metrics that evaluate how the number of bit writes are reduced, memory lifetimes increased, energy consumption reduced and storage overhead minimized.

We use SniperSim [13] to run workloads and generate memory traces of applications from the SPEC2006 benchmark suite [77]. Table 6.1 describes the parameters used in the simulated system.

---

88
Table 6.1: The system specifications of our modeled system.

<table>
<thead>
<tr>
<th>Processor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>X86</td>
</tr>
<tr>
<td>CMP</td>
<td>4-core, out-of-order, 2.66GHz</td>
</tr>
<tr>
<td>Interconnect Network</td>
<td>Bus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>On-chip Cache Hierarchy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>32KB/64B/8 way, private</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>256KB/64B/8 way, private</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>8MB/64B/16 way, shared</td>
</tr>
<tr>
<td>Store Policy</td>
<td>Write-Back</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PCM Main memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Capacity</td>
<td>4GB (4KB pages, and 64B rows)</td>
</tr>
<tr>
<td>Configuration</td>
<td>4 memory controller</td>
</tr>
<tr>
<td></td>
<td>1 rank/DIMM, 8 devices/rank,</td>
</tr>
<tr>
<td></td>
<td>1 chip for ECC, ECP [72] error correction</td>
</tr>
<tr>
<td>Endurance</td>
<td>Mean: $10^8$ writes, Variance: 0.25</td>
</tr>
</tbody>
</table>

Our baseline architecture, similar to Figure 6.3a, is a 4-core out-of-order multiprocessor system that includes three levels of on-chip cache in the memory hierarchy. The benchmarks are executed in rate mode [60], where all four cores execute the same benchmark. On an eviction from the LLC, write accesses are captured. The first 2 billion instructions for each benchmark are used to warm up on-chip caches, and then 10 million references to main memory (LLC misses, both reads and writes) are captured. Note that 10 million references normally corresponds to several billions of instruction execution for each benchmark.

6.2.1 Nacre Analysis: Segment Size

The size of each segment has a profound impact on the effectiveness of Nacre. However, using a finer granularity increases the storage overhead. As we discussed in Section 6.1.2, the storage overhead of Nacre is $M \times \lceil \log_2 N \rceil$. By modifying the segment size, we are altering the $M$ parameter, which has a linear relationship with the storage overhead. Figure 6.5 shows the impact of the segment size on the number of bit flips. The results are obtained by using four segment counters. For some of
the workloads, such as astar and gcc, results vary significantly as we change the segment size from 2 bytes to 8 bytes, almost doubling the number of bit flips. On the other end of spectrum, there are caculix and wrt, which are fairly insensitive to the segment size. On average, 17%, 21%, and 26% bit flips are observed using 2, 4, and 8 byte segments, respectively. Nacre using 2-byte segments increases bit flips modestly (17%), as compared to an unencrypted memory with 11.8% bit flips on average.

6.2.2 Nacre Analysis: Number of Counters

The storage overhead of Nacre is logarithmically proportional to the number of segment counters. We explore the impact of the number of segment counters on Nacre performance when the segment size is 2 bytes in Figure 6.6. Caculix and wrt show marginal improvements using more segment counters, since they experience 40% more bit flips versus using unencrypted memory. On
the other hand, increasing the number of segment counters can result in more than a 2x reduction in bit flips in some benchmarks, such as libquantum and sjeng. On average, Nacre with 2-5 segment counters leads to 25%, 19%, 17% and 16% fewer bit flips, respectively. Note that Nacre only increases bit flips by 5% when using 2-byte segments and 5 counters, as compared to the number of bit flips in unencrypted memory.

6.2.3 Comparison with Competing Approaches

6.2.3.1 Bit Flips Per Writes

For comparison, we model DEUCE [92] and BLE [48], two state-of-the-art counter-mode encryption techniques. BLE works at a granularity of 16 bytes, while DEUCE works at a two-byte granularity. A 3-bit local counter is assigned per block in BLE, and a write limit of 32 times is used as the epoch interval in the DEUCE implementation. We use four 3-bit segment counters with two byte segments as our default configuration for Nacre. We provide the percentage of bit flips per write for different schemes, along with results when using unencrypted memory in Figure 6.7.

For libquantum, the increase in the number of bit flips, as compared to unencrypted memory, can be more than 7x when using BLE and DEUCE, while Nacre increases bit flips by 2.8x for this benchmark. Nacre always outperforms DEUCE and BLE. DEUCE, in all the cases but sjeng, outperforms BLE. DEUCE does better when the data modifications are in the same segments, while in sjeng, the modified segment frequently changes on each writeback. The benefit of using Nacre over DEUCE is significant (more than 70% fewer bit flips) in libquantum, mcf, astar, sjeng, and gobmk benchmarks. Overall, the average number of bit flips for unencrypted memory, Nacre, DEUCE, and BLE are 11.84%, 17.46%, 26.27%, and 32.13%, respectively.

6.2.3.2 Energy Consumption

Increasing the number of bit writes imposed by an encryption algorithm has a tremendous impact on the PCM write energy. Adopting the energy model and power parameters employed by Zhou et al. [95], we take into account the energy consumed for writing each bit. Equation (6.3) shows the overall energy of the PCM memory.

\[ E_{PCM} = N_{read}E_{read} + N_{write}E_{write} \] (6.3)
CHAPTER 6. DURABLE, SECURE AND ENERGY-EFFICIENT NON-VOLATILE MEMORY

Figure 6.7: Average number of modified bits per write with various encryption schemes. Unencrypted memory, Nacre, DEUCE, and BLE experience 11.84%, 17.46%, 26.27%, and 32.13% bit flips, respectively.

where $N_{\text{read}}$ and $N_{\text{write}}$ are the number of reads and writes, respectively. $E_{\text{read}}$ are generated to read a 64B block from PCM, which is approximately equal to 1.075 nJ [95]. The write energy in PCM, assuming DCW is in place, can be modeled as in Equation (6.4) [95].

$$E_{\text{write}} = E_{\text{fixed}} + E_{\text{read}} + E_{\text{bitChange}}$$  \hspace{1cm} (6.4)

where $E_{\text{read}}$ is the energy consumed for the read access and placing the old data into a memory buffer, $E_{\text{fixed}}$ is the fixed energy consumed for write decoding, row selection, data comparison (using XNOR gates), etc., and finally, $E_{\text{bitChange}}$ takes into account the actual cell writes for a block of data. $E_{\text{fixed}}$ is 4.1 nJ per access [95]. $E_{\text{bitChange}}$ depends on the frequency of writing ones and zeros and is written as (6.5).

$$E_{\text{bitChange}} = N_{0 \rightarrow 1}E_{0 \rightarrow 1} + N_{1 \rightarrow 0}E_{1 \rightarrow 0}$$  \hspace{1cm} (6.5)

where $N_{0 \rightarrow 1}$ and $N_{1 \rightarrow 0}$ is the number of bit flips from 0 to 1 and vice-versa, respectively. $E_{0 \rightarrow 1}$, and $E_{1 \rightarrow 0}$ are the amounts of energy consumed when writing a ‘1’ or ‘0’, which are 0.0268 nJ and 0.013733 nJ, respectively [95]. With respect to the read and write frequency of each workload, the energy overhead of each encryption scheme is presented in Figure 6.8. All of the results are normalized versus unencrypted memory (red dashed line). The highest energy overhead is found in libquantum, which has the lowest number of bit writes among the benchmarks. For this workload, Nacre adds 15% energy overhead, while DEUCE and BLE increase energy consumption by 30% and 36%, respectively, as compared to unencrypted memory. Nacre induces small energy overheads in most cases. On average, 6%, 13%, and 17% more energy is consumed in Nacre, DEUCE, and BLE, respectively, versus an unencrypted memory.
6.2.3.3 Memory Lifetime

We employ the same methodology used in Schecter et al. [72], to model the lifetime of PCM. The lifetime of memory cells is assumed to follow a normal distribution, with a mean $\mu$ of $10^8$, and with a coefficient variation of 0.25. Similar to previous work [72, 84], we assume perfect wear leveling across the memory blocks, which is aligned with wear-leveling schemes previously proposed for PCMs [65].

We assume ECP [72] is used for error recovery. Whenever ECP cannot correct the faults in a PCM block, the entire page containing the memory block is removed from the accessible memory address space. Figure 6.9 plots the average lifetime of the memory system for different schemes. The metric of lifetime, which captures the percent of surviving pages with respect to the writes in pages, is independent of the size of the memory and is the common metric used for lifetime analysis [40, 72].

Fully encrypting memory lines results in a more than a 4x reduction in memory lifetime, on average. The reduction in lifetime of Nacre is 32% as compared to unencrypted memory. PCM lifetimes are reduced by 56% and 63% when DEUCE and BLE are used, respectively.

6.2.3.4 Hardware Overhead

Employing each scheme to reduce bit flips necessitates a few changes in the system. For example, in all schemes, the LLC needs to be modified to detect changes in the contents of a cache line. We would also need to add a few bits to show the modifications in segments. BLE only needs 4 bits to maintain the modifications in one cache line, while the default configuration of DEUCE and Nacre need 32 bits, since they work at a finer granularity.
DEUCE and Nacre require more AES blocks, as compared to BLE. Mathew et al. [58] designed an optimized and energy-efficient hardware implementation of AES for the 22 nm node. The AES unit area overhead is $\approx 0.02 \text{ mm}^2$, and introduces 216 cycles for decryption, while only consumes 13 mW power. A 4-core Intel Haswell processor at 22 nm technology has a die size of 260 $\text{ mm}^2$ [49], therefore, the AES unit area overhead (0.007%) is negligible in practice, even when multiple AES units are used.

Nacre and DEUCE schemes need to store a few bits as metadata, while BLE does not impose any extra storage overhead. As we discussed in Section 6.1.2, the ECC chip allows us to store up to 64 bits of metadata per memory line. We can re-purpose the ECC chip for storing mask metadata if a memory line is error free. In Table 6.2, the storage overhead of different schemes, as well as various configurations of Nacre, are provided.

6.2.3.5 Performance Overhead

State-of-the-art memory encryption schemes exploit counter-mode encryption, allowing OTP generation and data fetches from memory to be performed in parallel, hiding the memory latency on reads. Thus, encrypted memory will not incur a performance loss as compared to unencrypted
## Table 6.2: Storage overhead for various schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Overhead(bit)</th>
<th>Overhead(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>DEUCE</td>
<td>32-bit</td>
<td>6.25%</td>
</tr>
<tr>
<td>Nacre(2,2)</td>
<td>32-bit</td>
<td>6.25%</td>
</tr>
<tr>
<td>Nacre(2,4)</td>
<td>64-bit</td>
<td>12.50%</td>
</tr>
<tr>
<td>Nacre(4,2)</td>
<td>16-bit</td>
<td>3.12%</td>
</tr>
<tr>
<td>Nacre(4,4)</td>
<td>32-bit</td>
<td>6.25%</td>
</tr>
<tr>
<td>Nacre(8,2)</td>
<td>8-bit</td>
<td>1.56%</td>
</tr>
<tr>
<td>Nacre(8,4)</td>
<td>16-bit</td>
<td>3.12%</td>
</tr>
</tbody>
</table>

memory. Nonetheless, if the counters are not available in the counter cache, the fetch accesses and the pad generation processes cannot be overlapped, and an extra memory access penalty is incurred.

In PCM technology, the ability of the device to program bits in parallel is constrained by the maximum allowed current dissipation. This power limit is mainly due to poor efficiency of the charge pump circuits [32], and typically leads to multiple write slots if the number of bit flips in a memory block is high. For instance, a PCM chip with 2-16 concurrent bit writes was produced in 90nm technology [51], and Samsung demonstrated a 20nm PCM chip that supports 128-256 concurrent bit writes [20]. Therefore, depending on the PCM chip and technology node, multiple write slots may be required for a PCM memory block.

Despite that the write requests are buffered and not on the critical path, servicing them efficiently is critical for alleviating the memory contention for reads and improving performance. Several techniques have been reported in the literature that have shown how to reduce the overhead of extra write slots in PCM [18, 55, 64, 93]. Nacre reduces the number of bit flips in an encrypted memory significantly, as compared to the rival schemes, thus decreasing the number of required write slots. As compared to using an unencrypted memory, Nacre increases the average number of bit flips from $\sim 12\%$ to $\sim 17\%$, incurring minimal performance overhead.

### 6.3 Summary of Nacre

To protect sensitive data stored in NVMs and prevent malicious attacks, encryption can be used. However, encryption algorithms increase the number of bit flips and significantly impact
memory lifetimes. In this chapter, we proposed Nacre, an architecture support to mitigate the negative side effects of encryption on NVMs, enhancing lifetimes, while improving energy-efficiency.

Nacre attempts to reduce extra bit writes imposed by encryption by selectively re-encrypting modified memory segments and recording the history of data modifications. We found that, on average, Nacre improves memory lifetimes by 53% (2.87x) as compared to state-of-the-art (full encryption) schemes. Compared to unencrypted memory, Nacre increases bit flips by only 11.84% to 17.46%, on average, while increasing memory energy consumption by only 6%. We believe Nacre bridges the lifetime gap between encrypted and unencrypted NVMs and helps pave a path to NVM adoption for main memories in the near future.
Chapter 7

Summary

In this chapter, we review the contributions of this thesis, and discuss directions for future work in this area.

7.1 Main Contributions

In this thesis we performed an in-depth exploration of the challenges associated with dependability of phase change memory (PCM). Particularly, we addressed reliability and security as two main aspects of dependability factors in memory systems. Our contributions include:

- PCM wear-out continues to be a barrier, impacting the potential of this novel technology to replace DRAM in future main memory. We proposed REMAP \[83\] to enhance reliability of PCM, extending the lifetime by leveraging all the metadata bits as replacements for faulty bits in data. However, REMAP requires an extra memory access in order to locate the faulty bits. To eliminate or reduce this overhead, we consider three optimizations: partitioning, fault location caching, and using REMAP as a second layer of defense. We show that by adopting REMAP, PCM lifetime is drastically improved, handling many faulty bits per each memory block, while not imposing significant performance overhead.

- Workload characterization provides fundamental insights that allow us to design effective PCM mechanisms to extend lifetimes. To this end, we completed a comprehensive workload characterization that captures write access patterns, while focusing on memory lifetime analysis. We proposed new metrics for workloads that help to evaluate memory lifetime. Using trace-driven simulation, we capture the data transferred between on-chip memory and main memory.
CHAPTER 7. SUMMARY

- We developed a framework for lifetime analysis due to write accesses at a bit-level granularity using real data. We not only capture the writes to memory blocks, but we also capture the contents and the number of bit flips incurred at each writeback, to drive lifetime analysis. Moreover, start-gap [65] is integrated into our framework as a low-cost and state-of-the-art wear-leveling mechanism to efficiently distribute writes at a block level. Our framework models process variation in memory cells and generates synthetic workloads for memory lifetime evaluation.

- Due to the non-uniformity in memory writes, and the variation in the manufacturing process, metadata is underutilized, and early block failures occur in resistive memories. We proposed block cooperation techniques, where other blocks come to the aid of failed blocks to extend their lives and extend a page’s lifetime [84]. Block cooperation is realized through metadata sharing in ECP. Metadata sharing can be performed at a single level, where one data block shares its unused metadata with another data block, or at multiple levels, where multiple data blocks can share their metadata together. In Aegis, block cooperation is realized through data layout reorganization, where the blocks with fewer faults can help failed blocks in order to bring a page back to life. Using single level (or multi-level) block cooperation, we can increase memory lifetimes by 28% (37%), and 8% (14%) on average, for ECP and Aegis, respectively.

- Write disturbance remains another obstacle that must be overcome before PCM can become the technology of choice for future high capacity memory systems. We proposed two cost-effective techniques to reduce the probability of write disturbance [82]. The first technique (DMPart) encodes the data written to PCM in order to eliminate consecutive zeros, and reduces the chance of producing vulnerable data patterns. The second technique (SWEX), detects exposed cells and overwrites them if the number of cells is below a selected threshold. Otherwise, SWEX only overwrites cells which do not cause domino writes. The proposed techniques reduce the number of extra writes caused by write disturbance. Our experimental results showed that DMPart and SWEX reduce the average number of writes up to 32% and 49%, respectively, while incurring minimal impact on the lifetime and energy consumption of PCM.

- To protect sensitive data stored in non-volatile memory and prevent malicious attacks, encryption is used. However, encryption algorithms increase the number of bit flips and significantly...
CHAPTER 7. SUMMARY

impact memory lifetimes. We proposed Nacre, an architecture support to mitigate the negative side effects of encryption on non-volatile memory, enhancing lifetimes, while improving energy-efficiency [81]. Nacre attempts to reduce extra bit writes imposed by encryption by selectively re-encrypting modified memory segments and recording the history of data modifications. Nacre improves memory lifetimes by 53% (2.87x) as compared to state-of-the-art (full encryption) schemes. Moreover, it only increases memory energy consumption by 6% as compared to unencrypted memory. We believe Nacre bridges the lifetime gap between encrypted and unencrypted non-volatile main memory.

7.2 Future Work

To continue this work, we envision a number of exciting directions:

1. Designing dependable Non-volatile Memory (NVM) systems requires optimization of a number of metrics of the memory system, that are many times in conflict. To design a robust, reliable, low energy and durable memory system, extra metadata is commonly used in standard DIMMs, which imposes storage and cost overheads to the system. Co-optimization for reliability and security in NVMs reduces the overhead of dependable memory systems significantly. Moreover, repurposing the components that are used for security in memory for reliability, or vice-versa, is the key for co-optimization and co-management of such designs at low cost.

2. Cross-layer approaches that exploit additional layers of abstraction in the system stack, such as the operating systems (OS) or the run-time environment, is another avenue to address dependability and energy-efficiency of NVMs. For instance, fault-tolerant data structures can be used at a software level to mitigate reliability of NVMs, or feedback from resiliency of algorithms (through compilers or programmer hints) can be fed into the memory DIMM to save energy by an approximate write at the circuit level (i.e., reducing the voltage for the write).

3. A hybrid memory system pairs DRAM with NVM technology to combine the benefits of both memories technologies. It allows us to enjoy the high bandwidth and low latency of current DRAM technology, while at the same time allows us to exploit the high capacity, energy-efficiency, and persistency of NVMs. A small DRAM can be configured as a cache for a large NVM, or it can be configured in a flat-addressable hybrid memory architecture. The
main challenge is managing the pages between two types of memory, dealing with the range of characteristics of different phases of an application. Mapping should maximize performance, but at the same time, reduce the memory writes to NVM to increase memory lifetime. Machine learning algorithms can be used for phase classification of applications to find any performance sensitivities of the different memory technologies and guide the OS to dynamically migrate pages between volatile and non-volatile portions of memory to maximize performance.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


[60] Prashant J Nair, Chiachen Chou, Bipin Rajendran, and Moinuddin K Qureshi. Reducing read latency of phase change memory via early read and turbo read. In High Performance Computer
BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


