University of California
Santa Barbara

A Statistical View of Architecture Design

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Zhaoxia Deng

12 2017

The Dissertation of Zhaoxia Deng is approved.

Diana Franklin (Committee Member)

Tim Sherwood (Committee Member)

Fred Chong (Advisor), Committee Chair

12 2017

A Statistical View of Architecture Design

To my family.

Acknowledgements

There are a number of people that I want to thank for their help, support, and guidance to make this dissertation possible. First and foremost, I would like to thank my advisor, Prof. Fred Chong, for introducing me to this interesting area of exploiting statistical techniques to understand and design complicated systems and architectures, as well as guiding me to explore many statistical methods. During the study, I learned the knowledge in both statistics and architecture, as well as the novel application of some fundamental theories. His broad experience in multiple fields often led me to a better perspective of the big picture. I am also very grateful for his patience in answering me many questions, inspiring me the out-of-box thinking when solving problems, teaching me the methodology of conducting experiments, and demonstrating how to improve presentation and writing skills. Also, I appreciate for all the financial support that let me study at the University of Chicago as a visiting student for the last two years while maintaining the graduate student status at the University of California, Santa Barbara.

I am very grateful to Prof. Alberto-Giovanni Busetto, Prof. Pradeep Sen, and Brian Drawert for spending a lot of time with us discussing intriguing ideas and novel applications. The discussion has brought many excitements and a lot of fun. The statistical techniques and experimental methodology I learned from them have also been very useful in the later projects.

I am also thankful to the rest of my committee members, Prof. Diana Franklin and Prof. Tim Sherwood, for helping me throughout the Ph.D. study, being on my thesis committee and giving me helpful feedback on the three pieces of work included in the dissertation. As a successful woman researcher, Prof. Diana Franklin has also been a role model and shown me great passion and encouragement to continue the research work.

In the end, I would like to thank the rest of my collaborators behind this dissertation, Prof. Stuart K. Kurtz, Prof. Ariel Feldman, Prof. Henry Hoffmann, Lunkai Zhang, and Nikita Mishra from the University of Chicago, for all the inspiration, encouragement, and help.

Curriculum Vitæ
Zhaoxia Deng

Education

2017	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2010	B.A. in Computer Science, Shandong University, China.

Publications

Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Trade-offs in NVMs
Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, Frederic T. Chong. The 50th International Symposium on Microarchitecture (Micro), 2017, Boston, MA.

Thermal-aware, heterogeneous materials for improved energy and reliability in 3D PCM architectures
Heba Saadeldeen, Zhaoxia Deng, Timothy Sherwood, Frederic T. Chong. The International Symposium on Memory Systems (MemSys), 2017, Washington, DC.

Lemonade from Lemons: Harnessing Device Wearout to Create Limited-Use Security Architectures
Zhaoxia Deng, Ariel Feldman, Stuart A. Kurtz, Frederic T. Chong. The 44th International Symposium on Computer Architecture (ISCA), 2017, Toronto, Canada.

Herniated Hash Tables: Exploiting Multi-Level Phase Change Memory for In-Place Data Expansion
Zhaoxia Deng, Lunkai Zhang, Diana Franklin and Frederic T. Chong. The International Symposium on Memory Systems (MemSys), 2015, Washington, DC.

Quantum Rotations: A Case Study in Static and Dynamic Machine-Code Generation for Quantum Computers
Daniel Kudrow, Kenneth Bier, Zhaoxia Deng, Diana Franklin, Yu Tomita, Kenneth R. Brown, and Frederic T. Chong. The 40th International Symposium on Computer Architecture(ISCA), 2013, Tel Aviv, Israel.

Abstract

A Statistical View of Architecture Design

by

Zhaoxia Deng

Computer architectures are becoming more and more complicated to meet the continuously increasing demand on performance, security and sustainability from applications. Many factors exist in the design and engineering space of various components and policies in the architectures, and it is not intuitive how these factors interact with each other and how they make impacts on the architecture behaviors. Seeking for the best architectures for specific applications and requirements automatically is even more challenging. Meanwhile, the architecture design need to deal with more and more non-determinism from lower level technologies. Emerging technologies exhibit statistical properties inherently, such as the wearout phenomenon in NEMs, PCM, ReRAM, etc. Due to the manufacturing and processing variations, there also exists variability among different devices or within the same device (e.g. different cells on the same memory chip). Hence, to better understand and control the architecture behaviors, we introduce the statistical perspective of architecture design: by specifying the architectural design goals and the desired statistical properties, we guide the architecture design with these statistical properties and exploit a series of techniques to achieve these properties.

In the first part of the thesis, we introduce *Herniated Hash Tables*. Our architectural design goal is that the hash table implementation is highly scalable in both storage efficiency and performance, while the desired statistical property is to achieve as good storage efficiency and performance as with uniform distributions given non-uniform distributions across hash buckets. *Herniated Hash Tables* exploit multi-level phase change memory (PCM) to in-place expand storage for each hash bucket to accommodate asymmetrically chained entries. The organization, coupled with an addressing and prefetching scheme, also improves performance significantly by creating more memory parallelism.

In the second part of the thesis, we introduce *Lemonade from Lemons, harnessing device wearout to create limited-use security architectures*. The architectural design goal is to create hardware security architectures that resist attacks by statistically enforcing an upper bound on hardware uses, and consequently attacks. The desired statistical property is that the system-level minimum and maximum uses can be guaranteed with high probabilities despite of device-level variability. We introduce techniques for architecturally controlling these bounds and explore the cost in area, energy and latency of using these techniques to achieve system-level usage targets given device-level wearout distributions.

In the third part of the thesis, we demonstrate *Memory Cocktail Therapy: A General, Learning-Based Framework to Optimize Dynamic Tradeoffs in NVMs*. Limited write endurance and long latencies remain the primary challenges of building practical memory systems from NVMs. Researchers have proposed a variety of architectural techniques to achieve different tradeoffs between lifetime, performance and energy efficiency; however, no individual technique can satisfy requirements for all applications and different objectives. Our architectural design goal is that NVM systems can achieve optimal tradeoffs for specific applications and objectives, and the statistical goal is that the selected NVM configuration is nearly optimal. *Memory Cocktail Therapy* uses machine learning techniques to model the architecture behaviors in terms of all the configurable parameters based on a small number of sample configurations. Then, it selects the optimal configuration according to user-defined objectives which leads to the desired tradeoff between performance, lifetime and energy efficiency.

Contents

Curriculum Vitae **vi**

Abstract **vii**

1 Introduction **1**
1.1 Motivation 2
1.2 Thesis statement 5
1.3 Thesis contribution 5
1.4 The outline of the thesis 8

2 Herniated Hash Tables: Exploiting Multi-Level Phase Change Memory for In-Place Data Expansion **9**
2.1 Introduction 9
2.2 Background of Multi-Level PCM 10
2.3 Herniated Hash Tables 12
2.4 Methodology 20
2.5 Results 26
2.6 Related Work 36
2.7 Chapter Summary 37

3 Lemonade from Lemons: Harnessing Device Wearout to Create Limited-Use Security Architectures **38**
3.1 Introduction 38
3.2 Device wearout model 40
3.3 Threat model 44
3.4 A Limited-use connection 44
3.5 A limited-use targeting system 58
3.6 Using device wearout to build one-time pads 60
3.7 Limitations 72
3.8 Related work 73
3.9 Chapter Summary 75

4 Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Tradeoffs in NVMs **77**
4.1 Introduction 77
4.2 Background 81
4.3 Case study 85
4.4 Memory Cocktail Therapy 89
4.5 Implementation 98
4.6 Results 103
4.7 Related Work 112
4.8 Chapter Summary 113

5 Future Work **115**
5.1 Hardware support for datastructures 115
5.2 Rate Limiting Security architectures 118
5.3 Statistical Inference and Optimization of NVM Systems 119

6 Conclusion **121**

Bibliography **123**

Chapter 1

Introduction

Today's computing systems are becoming tremendously complex to meet various kinds of demands from applications. In the big data era, applications generate massive data at a speed of TB per second. The large-scale data continuously demands for faster processing capability, larger memory capacity and higher throughput, etc. Furthermore, many new types of applications emerge, such as IoT devices, self-driving vehicles, security systems, automated targeting systems, etc. Each system seeks for its specific sweet-spot between performance, energy efficiency, accuracy, security, reliability, etc.

To meet all these changing demands from applications, the computing systems have also evolved from generation to generation. On the one hand, architects strive to scale up the system performance by intellectually organizing various components and carefully engineering each component and policy, despite of the diminishing impact of Moore's Law and many challenges in the scaling of traditional DRAM systems. However, the increasing complexity made it extremely difficult to reason about and design these architectures. On the other hand, more and more complexity has been pushed to the software stack, which drives the birth of many kinds of cluster computing frameworks, including those tailored for large scale machine learning problems. To bring the simplicity back to upper-level software systems, architects have never stopped exploring the emerging hardware technologies to either solve problems in traditional architectures or reduce their inherent complexity .

Many emerging technologies have shown promising characteristics to architect future computing systems. Non-volatile memories (NVMs) exhibit high density, low static power, and persistence, which are promising to solve the DRAM scaling issues in traditional architectures. For example, phase change memory (PCM) has been shown possible to work as a scalable alternative of the traditional DRAM system. Memristors have also been used to architect components such as branch predictors and cache directories. In addition to NVMs, nano-electromechanical devices (NEMs) and molecular devices have smaller scales of physical dimensions and high tolerance to different temperatures, radiation conditions and electric fields so that they could be deployed in harsh environments.

This thesis aims to explore new architectures with these emerging technologies, from a statistical perspective. In the following sections, we first discuss the new challenges in the architecture design with emerging technologies. Also, we rethink the architecture design methodology and exploit statistical techniques to handle these challenges.

1.1 Motivation

1.1.1 New challenges in architecture design with emerging technologies

There are many challenges associated with building practical systems from emerging technologies. This thesis focuses on exploring the potential of phase change memory (PCM) and NEMs switches. We discuss three of the most common challenges in the following sections.

Longer read and write latency in PCM

In multi-level PCM, a single PCM cell can be used to store multiple bits in an asymmetric manner. The deeper bits (less-significant bits) need more time to be read out and much more time to be updated than shallower bits (more-significant bits). For read operations, the relationship between the performance penalty and the number of bits read out is about exponential. However, it is even worse for write operations. Qureshi et al. proposed schemes such as write cancellation and write pausing to avoid the delay of response to the following read requests. Nevertheless, frequent write cancellation operations may hurt the performance and lifetime of NVM systems.

Limited write endurance

It is common that emerging NVM technologies usually have limited write endurance. For example, the typical write endurance of PCM cells is about 10^8. If without special treatment, the future memory systems from emerging technologies will have short lifetime as (some of) the memory cells will be worn out soon. As a result, special mechanisms (eg., wear leveling and/or wear limiting) must be used to guarantee the lifetime of NVM memories.

Device failures of NEMs switches

Even worse than the limited write endurance of PCM cells, NEMs switches suffer from device failures after tens or hundreds of cycles. Representative NEMS contact switches are composed of a movable active element and an opposing electrode which interact by both electrical and mechanical forces to open and close the switch. Generally speaking, any kind of electrical and mechanical aging, adhesion, fracture or burnout in the active element or electrode are potentially responsible for the failures of NEMS switches.

1.1.2 Why is the statistical perspective helpful?

The quantitative approach has been the classical methodology of architecture design. However, simply relying on quantitative measurements seem insufficient to handle the tremendous complexity of recent architectures and the new challenges exposed from emerging technologies. In this thesis, we propose to specify statistical properties at the interfaces between applications, architectures and devices. The statistical interface will be necessary for two reasons: one arises from the large-scale, fine-grained configuration space; another from the non-determinism of device level technologies.

Large-scale, fine-grained configuration space

Due to various types of applications and computing environments, each system needs to satisfy specific QoS requirements while seeking for its sweet-spot among performance, energy efficiency, accuracy, manufacturing cost, etc. Meanwhile, recent architectures usually consist of many components and complicated policies to coordinate these components. It is not intuitive how to determine the best configuration for each specific application and objective from the large-scale, fine-grained configuration space. An expressive abstract between the hardware and applications is demanded so that the capacity of hardware can be utilized maximally and the optimization of configurations can be done adaptively.

Non-determinism of device level technologies

The wearout phenomenon is commonly seen in emerging technologies such as NEMs, PCM, ReRAM, etc, which means that the device can only work for a limited number of times. As most researchers try to mitigate the wearout problem, we take a contrarian view and exploit the wearout to create physically-enforced security architectures. The challenging issue, however, is that it is usually not deterministic when the device will stop functioning. Furthermore, manufacturing and processing variations can result in significant variations among devices or cells on the same device. Therefore, to treat the non-determinism of device variations, we exploit probabilistic modeling of the underlying device behaviors and exploit a series of techniques to provide statistical guarantees on the architectural behaviors.

1.2 Thesis statement

In this thesis, we propose the statistical architecture design methodology, to treat the above challenges in emerging technologies.

By specifying the architectural design goals and the desired statistical properties, we guide the architecture design with these statistical properties and exploit a series of techniques to achieve these properties.

In general, we first formalize the statistical interface between applications, architectures and devices. Then, the probabilistic modeling of the underlying devices is used to handle the device variability. Also, machine learning techniques are helpful to dynamically select the optimal configuration for specific applications and targets.

1.3 Thesis contribution

In this thesis, we first explore the novel architecture design with emerging technologies with the following three cases. Each of the three architectures resolve some of these challenges in building new systems from emerging technologies. Furthermore, we investigate the statistical architecture design methodology with these cases. The contributions of each architecture are listed in the following sections.

1.3.1 Herniated Hash Tables: Exploiting Multi-Level Phase Change Memory for In-Place Data Expansion

Hash tables are a commonly used data structure used in many algorithms and applications. As applications and data scale, the efficient implementation of hash tables becomes increasingly important and challenging. In particular, memory capacity becomes increasingly important and entries can become asymmetrically chained across hash buckets. This chaining prevents two forms of parallelism: memory-level parallelism (allowing multiple prefetch requests to overlap) and memory-computation parallelism (allowing computation to overlap memory operations). We propose, herniated hash tables, a technique that exploits multi-level phase change memory (PCM) storage to expand storage at each hash bucket and increase parallelism without increasing physical space. The technique works by increasing the number of bits stored within the same resistance range of an individual PCM cell. We pack more data into the same bit by decreasing noise margins, and we pay for this higher density with higher latency reads and writes that resolve the more accurate resistance values. Furthermore, our organization, coupled with an addressing and prefetching scheme, increases memory parallelism of the herniated datastructure. We simulate our system with a variety of hash table applications and evaluate the density and performance benefits in comparison to a number of baseline systems. Compared with conventional chained hash tables on single-level PCM, herniated hash tables can achieve 4.8x density on a 4-level PCM while achieving up to 67% performance improvement.

1.3.2 Lemonade from Lemons: Harnessing Device Wearout to Create Limited-Use Security Architectures

Most architectures are designed to mitigate the usually undesirable phenomenon of device wearout. We take a contrarian view and harness this phenomenon to create hardware security mechanisms that resist attacks by statistically enforcing an upper bound on hardware uses, and consequently attacks. For example, let us assume that a user may log into a smartphone a maximum of 50 times a day for 5 years, resulting in approximately 91,250 legitimate uses. If we assume at least 8-character passwords and we require login (and retrieval of the storage decryption key) to traverse hardware that wears out in 91,250 uses, then an adversary has a negligible chance of successful brute-force attack before the hardware wears out, even assuming real-world password cracking by professionals. M-way replication of our hardware and periodic re-encryption of storage can increase the daily usage bound by a factor of M. The key challenge is to achieve practical statistical bounds on both minimum and maximum uses for an architecture, given that individual devices can vary widely in wearout characteristics. We introduce techniques for architecturally controlling these bounds and perform a design space exploration for three use cases: a limiteduse connection, a limited-use targeting system and one-time pads. These techniques include decision trees, parallel structures, Shamirs secret-sharing mechanism, Reed-Solomon codes, and module replication. We explore the cost in area, energy and latency of using these techniques to achieve system-level usage targets given device-level wearout distributions. With redundant encoding, for example, we can improve exponential sensitivity to device lifetime variation to linear sensitivity, reducing the total number of NEMS devices by 4 orders of magnitude to about 0.8 million for limited-use connections (compared with 4 billion if without redundant encoding).

1.3.3 Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Tradeoffs in NVMs

Non-volatile memories (NVMs) have attracted significant interest recently due to their high-density, low static power, and persistence. There are, however, several challenges associated with building practical systems from NVMs, including limited write endurance and long latencies. Researchers have proposed a variety of architectural techniques which can achieve different tradeoffs between lifetime, performance and energy efficiency; however, no individual technique can satisfy requirements for all applications and different objectives. Hence, we propose Memory Cocktail Therapy (MCT), a general, learning-based framework that adaptively chooses the best techniques for the current application and objectives. Specifically, MCT performs four procedures to adapt the techniques to various scenarios. First,MCTformulates a high-dimensional configuration space from all different combinations of techniques. Second, MCT selects primary features from the configuration space with lasso regularization. Third, MCT estimates lifetime, performance and energy consumption using lightweight online predictors (eg. quadratic regression and gradient boosting) and a small set of configurations guided by the selected features. Finally, given the estimation of all configurations, MCT selects the optimal configuration based on the user-defined objectives. As a proof of concept, we test MCTs ability to guarantee different lifetime targets and achieve 95% of maximum performance, while minimizing energy consumption. We find that MCT improves performance by 9.24% and reduces energy by 7.95% compared to the best static configuration. Moreover, the performance of MCT is 94.49% of the ideal configuration with only 5.3% more energy consumption.

1.4 The outline of the thesis

The rest of the thesis will present three architectures with more details, in Chapter 2, 3, 4, respectively. Then we will discuss the future work that follows the three architectures in Chapter 5. In the end, we conclude the thesis in Chapter 6.

Chapter 2

Herniated Hash Tables: Exploiting Multi-Level Phase Change Memory for In-Place Data Expansion

2.1 Introduction

Compared with DRAM, *phase-change memory* (PCM) shows a number of promising attributes, including higher density, non-volatility and low static power consumption. Therefore, it is considered a possible substitute for the DRAM technique. Another interesting attribute of PCM is that a single PCM cell can be used to store multiple bits in an asymmetric manner. The deeper bits (less-significant bits) need more time to be read out and much more time to be updated than shallower bits (more-significant bits). However, for the long read latency, it is possible to develop a prefetching mechanism that manages to prefetch out the deep-level data from PCM before their usage with a step-wise multi-level read operation. For the long write latency, Qureshi et al. [1] have proposed schemes such as write cancellation and write pausing to avoid the delay of response to the following read requests. Therefore, the overall system needs not to suffer from the long access latency of deep-level data.

Conventional hash tables usually use chained structures to save memory space. The addresses of entries are allocated dynamically and distributed randomly in memory, which prevents prefetching and instruction-level parallelism. Moreover, with multi-level PCM, the performance of conventional hash tables will decrease dramatically because of the high read and write latency. In this chapter, we introduce the *herniated hash table*, which exploits multi-level PCM and provides dense and efficient storage with support of dynamic in-place expansion and non-uniform growth of the hash tables. By designing an addressing scheme for the multi-level PCM and using multi-level PCM aware hash table structures and prefetching schemes, the herniated hash table performs better than the conventional hash table design from the following two perspectives: On one hand, the herniated hash table yields similar performance with less storage compared with the conventional hash table on single-level PCM; On the other hand, the herniated hash table achieves much better performance with similar storage overhead compared with the conventional hash table on multi-level PCM.

The rest of the chapter is organized as follows. We first provide background information on multi-level PCM and its operations in Section 2.2. Then, Section 2.3 presents our herniated hash table design. Section 2.4 describes our experimental methodology, and Section 2.5 presents our results. Finally, Section 2.6 discusses related work, and Section 2.7 draws our conclusions.

2.2 Background of Multi-Level PCM

Phase-change memory (PCM) is one type of resistive memory, the resistance of whose cells can be changed in a wide range. An analog-to-digital circuit (ADC) reads the resistance as a digital value. Depending on the ADC setting, the same resistance of a PCM cell could be interpreted as a different number of bits. Alibart et al. [2] showed that a single resistive

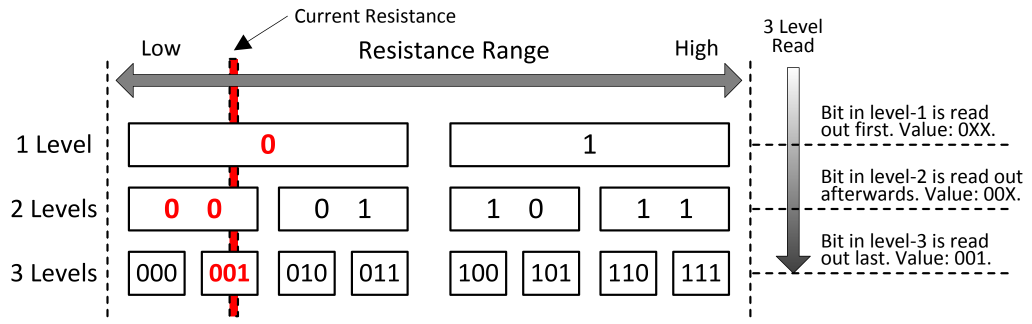

Current Resistance
Low
Resistance Range
High
3 Level Read
1 Level
0
1
2 Levels
0 0
0 1
1 0
1 1
3 Levels
000
001
010
011
100
101
110
111
Bit in level-1 is read out first. Value: 0XX.
Bit in level-2 is read out afterwards. Value: 00X.
Bit in level-3 is read out last. Value: 001.

Figure 2.1: **A multi-level PCM cell read; the lower-level bits need to be read out first before the deepest-level bit is read out.**

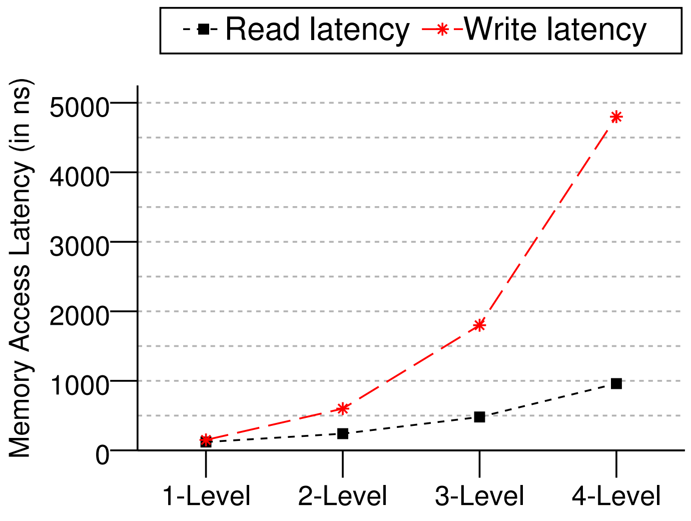


Figure 2.2: **Latency to access different PCM levels.**

memory cell can be used to store up to seven bits of data. Figure 2.1 shows an example of a multi-level PCM cell: the same resistance of the cell can be interpreted as b0 when the ADC is using one-level accuracy and as b00 and b001, respectively, when the ADC is using two-level and three-level accuracy. This precision comes with a cost. Read latency increases exponentially with the number of bits stored in the PCM cell and write latency is even higher than that. However, as shown in Figure 2.1, multi-level PCM read can be operated in a step-wise approximate manner: that is, after reading a shallow bit of a PCM cell, the read operation can be continued to further read out its deeper_level bits. Therefore, if the data are carefully organized in multi-level PCM, we can use this step-wise read attribute to develop an efficient prefetching mechanism.

Although write latency of deep-level PCM accesses is also very high, we assume that our target applications have much more read accesses than write accesses to the hash tables. Moreover, as our scheme enables both instruction-level parallelism and memory-computation parallelism, the write latency (which is usually caused by the write back requests from the last level cache) doesn't have significant effects on the critical path of the processor's pipeline. If the write latency is too long that the following read requests are affected, write cancellation and write pausing techniques [1] will help by prioritizing the following read requests to remove potential pipeline stalls. We use the conservative configuration with the write cancellation technique in all of our systems that run on the multi-level PCM. Better combinations of write cancellation and write pausing are possible according to [1]. We'll also discuss the case without write cancellation in Section 2.5.

In this work, similar to previous work [3], we have an exponential latency model for multi-level PCM read accesses (the exponential base is 2.0) and a higher latency model for multi-level PCM write accesses: as shown in Figure 2.2, a PCM read takes 120 ns, 240 ns, 480 ns, and 960 ns respectively, and a PCM write takes 150 ns, 600 ns, 1800 ns, 4800 ns respectively for one, two, three, and four levels.

2.3 Herniated Hash Tables

In this section, we firstly explain the inefficiency of the conventional hash table design on multi-level PCM and present the design of the herniated hash table (HHT), with emphasis on the HHT's differences from a conventional hash table on the same PCM system. Then we

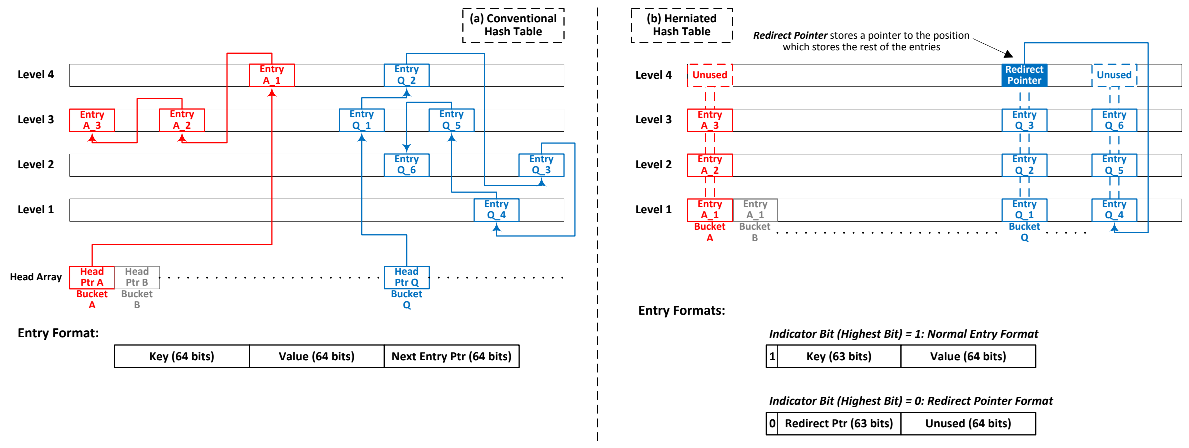


Figure 2.3: **Hash table organization on multi-level PCM. We present two cases: a conventional linked list-based hash table and the herniated hash table.**

introduce a new addressing scheme for the multi-level PCM system since traditional virtual to physical address mapping schemes are not efficient in multi-level PCM. Based on that, we design multiple prefetching schemes to improve the performance of memory accesses.

The conventional hash table is implemented with an array of linked lists. The length of the array is the number of buckets. Each array entry is an address that points to the linked list of hash entries that fall into the same bucket. The single-level PCM system behaves similarly as traditional DRAM. Without loss of generality, we assume the single-level PCM capacity is 1 GB. The multi-level PCM system is assumed to be *four* levels, with 0–1 GB on the first level, 1–2 GB on the second level, 2–3 GB on the third level, and 3–4 GB on the fourth level. Multi-level PCM systems with more levels will also be analyzed in the sensitivity studies.

2.3.1 The Basic Herniated Hash Table Design

Figure 2.3 shows a snapshot of the organizations of both the conventional hash table and the HHT. Both of them contain N buckets.

The conventional hash table, as shown in Figure 2.3(a), starts with an array of N pointers (i.e., *head pointers*), and each head pointer points to the first valid entry of the corresponding

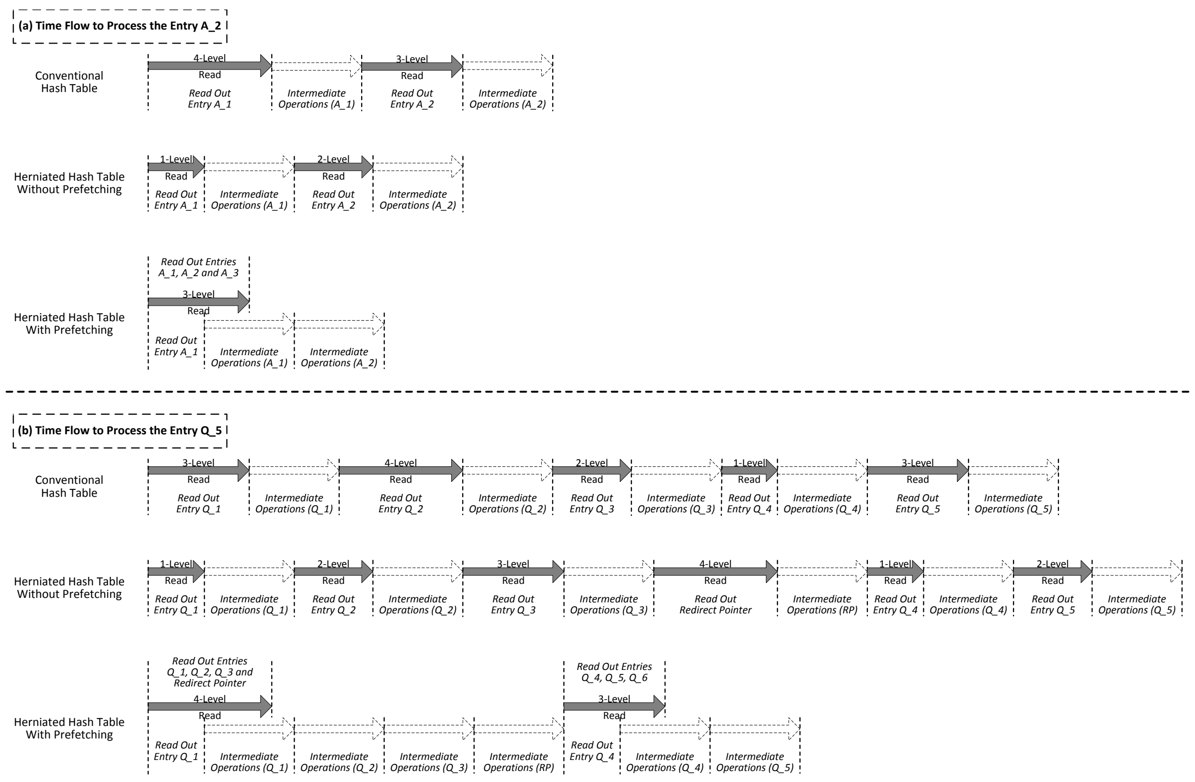

(a) Time Flow to Process the Entry A_2
Conventional Hash Table
Herniated Hash Table Without Prefetching
Herniated Hash Table With Prefetching
4-Level Read
3-Level Read
2-Level Read
1-Level Read
Read Out Entry A_1
Intermediate Operations (A_1)
Read Out Entry A_2
Intermediate Operations (A_2)
Read Out Entries A_1, A_2 and A_3
(b) Time Flow to Process the Entry Q_5
Read Out Entry Q_1
Intermediate Operations (Q_1)
Read Out Entry Q_2
Intermediate Operations (Q_2)
Read Out Entry Q_3
Intermediate Operations (Q_3)
Read Out Entry Q_4
Intermediate Operations (Q_4)
Read Out Entry Q_5
Intermediate Operations (Q_5)
Read Out Redirect Pointer
Intermediate Operations (RP)
Read Out Entries Q_1, Q_2, Q_3 and Redirect Pointer
Read Out Entries Q_4, Q_5, Q_6

Figure 2.4: **The timing of accessing different hash entries in Figure 2.3.**

hash bucket. Each conventional hash table entry contains three fields: *key*, which identifies the entry; *value*, which contains the information for the entry; and *next entry ptr*, which points to the next entry in the same bucket. This conventional implementation is not very efficient on a multi-level PCM. The main problem is that, since it is not multi-level PCM-aware, both the head pointers and entries can be placed randomly at any level in PCM, and therefore may need a long latency to access if they happen to be in deep levels. Another intrinsic problem of the conventional hash table is that it uses pointers to link the entries and the addresses of entries are dynamically allocated and randomly distributed in the physical memory address space. It is hard to either parallelize the computation and data retrieving because we don't know the address of the next entry beforehand, or use a prefetching mechanism to retrieve the entries faster due to the lack of spacial locality in memory.

In Figure 2.3(b), we show the organization of the HHT. At first, the HHT only uses the first level of the PCM, and it initially contains an array of *N* hash entries (i.e., the first hash entry of each bucket). Whenever possible, the HHT always puts the entries from the same bucket in the same PCM cells but in different levels (e.g., entries A_1 to A_3). If the current cells have already reached their level limit (in our case, four levels), new PCM cells will be allocated to store extra hash entries. The deepest level of the previous PCM cells will be transferred to a *redirect pointer*, which points to the newly allocated space at the first PCM level (e.g., entry Q_4 in the figure). This data organization creates spacial locality in memory and removes most of the links that blocks instruction-level parallelism. In general, compared with the conventional design, the benefits of the HHT are three-fold:

- **More Compact Hash Table Structure.** As shown in Figure 2.3, each entry in the HHT has only two fields, *key* and *value*, and it does not need a pointer field (like *next entry ptr* in a conventional hash entry). Moreover, the HHT does not need a separate array of *head pointers*, which is also needed in the conventional hash table. The HHT, however, has two storage drawbacks. First, it has to pre-allocate a hash entry for every bucket (even if a bucket is never used at all). Second, if there are too many entries in one bucket, then additional storage is needed for *redirect pointers*. However, we shall see in Section 2.5 that the storage merits of HHTs usually outweigh their drawbacks.

- **Shorter Latency to Access Buckets with Few Entries.** HHTs always insert a new hash table entry into the shallowest unused level. Therefore, for a hash bucket with few entries (i.e., 1-2 entries), only the shallow PCM levels are used. As a result, accessing these entries in these buckets will have a relatively short latency. Figure 2.4(a) shows such a situation: in the conventional hash table, entries A_1 and A_2 are buried in deeper levels (levels 4 and 3, respectively), whereas, in the HHT, these two entries are deliberately placed in the shallowest levels (levels 1 and 2, respectively). As a result, it takes less time to access entry A_2 in the HHT.

Note that, when accessing an entry in a bucket with many entries, the HHT without prefetching may take a longer time than the conventional hash table, as shown in Figure 2.4(b). In this case the HHT also needs to use deeper PCM levels, and it takes a long time to access the entries. Additionally, the HHT needs extra time to access redirect pointers. We improve the system performance in this situation with prefetching, which will be explained later.

- **Ability to Prefetch Entries.** In a hash table lookup function, to find the matched entry, the function needs to traverse through the hash entries of the corresponding bucket. In a conventional hash table, since there is no special relationship among the addresses of successive entries (because they are linked by pointers), prefetching can not be accomplished without complex pointer-tracing hardware. The HHT, however, stores several entries contiguously for each bucket, which enables prefetching. With step-wise reading, we can get multiple entries ready with exponential latency. The prefetching is very effective for hash tables because the access pattern tends to be traversing all the entries in the same bucket. We will discuss the prefetching schemes for the HHT in detail in Section 2.3.3.

2.3.2 Addressing HHTs

A critical challenge in exploiting multi-level PCM lies in how the data will be addressed and stored in the cache hierarchy. To access data at different levels in the same PCM cells, each level is potentially to be addressed as a separate physical page for a given page of physical PCM cells. Correspondingly, four-times (for four-level PCM) of the virtual address space will be utilized to represent all entries. Traditional virtual to physical address translation schemes [4][5][6][7] include multi-level forward page tables [8][9], inverted page tables [10] or hashed page tables [11] for memory efficient mapping between virtual addresses and physical addresses. However, when both the utilized physical and virtual address space are scaled, the existing schemes will all suffer from proportional increase in storage for the page tables.

HHTs exploit a multi-level PCM aware addressing scheme. The HHT tries to store the hash entries belonging to the same bucket in different levels of the same PCM cells. In addressing, this corresponds to different virtual pages with the same offset. Only the most significant address bits differ in addresses for different virtual pages. In this manner, successive HHT entries in the same bucket can be directly addressed by software by using the corresponding addresses in a sequence of physical pages. We don't need to increase storage for the page table since the addressing of different virtual pages for the same hash bucket follows a sequential pattern.

This scheme requires us to pay for potential hash table growth in terms of physical address, but allows us to defer allocating physical space in the HHT as needed for each hash table entry. For example, an HHT based upon four-level PCM would be addressed by four-times the physical addresses, but it would begin with only one-time the actual storage and grow as needed as the number of hash entries increases in one bucket.

An important advantage of this physical addressing scheme is that it allows multi-level PCM to be seamlessly integrated into the hash hierarchy. Data at multiple levels can be easily fetched (and prefetched) into caches. To avoid cache conflicts, we sequentially add a shift: *the cache block size* to the addresses of different virtual pages for the same hash bucket so that entries in this bucket will fall into different cache sets.

2.3.3 Prefetching Schemes for Herniated Hash Tables

The prefetching schemes for the HHT are based on two observations. First, the hash entries in the same bucket tend to be placed in the same PCM cells. Second, to read out a deep entry, the shallower entries in the same PCM cells (e.g., entries A_1 and A_2) need to be read out first.

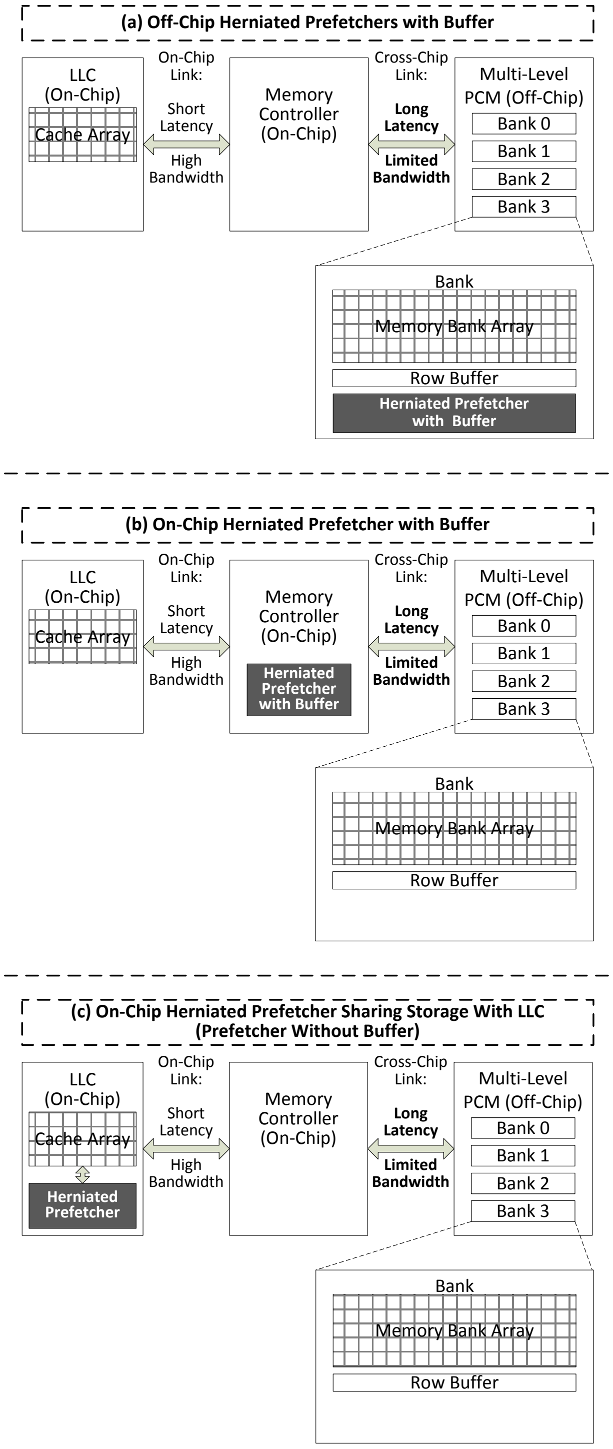


Figure 2.5: **Different prefetching schemes for the herniated hash table**

Therefore, the basic idea of prefetching schemes in the HHT is that, when accessing a shallow-level entry, we perform a deep-level read that reads out all the entries on the corresponding memory cells and buffers them for future use. Figure 2.4 illustrates the timing diagrams for such a case: when the program accesses entry A_1 (which is on the first level), instead of operating a single-level PCM read, the memory controller issues a three-level PCM read to read out all the successive used entries on the same memory cells (entries A_1, A_2, and A_3). Therefore, when the program accesses entry A_2 later, there is no longer a need to operate a PCM read. Note that, since the data in the multi-level PCM are read out in a step-wise manner, entry A_1 is read out with the same latency (level_one read latency) irrespective of the total number of bits stored in the same cells. That is, when using prefetching, there is no performance penalty to read out shallow-level entries.

There are multiple trade-offs in implementing a practical prefetching mechanism for the HHT (named as *herniated prefetcher*). Here we present the two most important ones:

- **On-Chip vs. Off-Chip.** In a modern processor, the memory controllers are usually implemented on-chip and the main memory (PCM in our case) is placed off-chip. The interface between these two components involves long latency and limited bandwidth. If the herniated prefetcher is implemented on-chip, as shown in Figures 2.5(b) and 2.5(c), the processor can get the prefetched data from the HHT with relatively low latency, but this will require higher memory bandwidth to transfer all the prefetched data from off-chip memory. If the prefetcher is implemented off-chip, as shown in Figures 2.5(a), the situation is just the opposite—there is no need for additional memory bandwidth, but the processor may suffer longer latency to access the prefetched data.

- **Buffering vs. Bufferless.** A prefetcher often comes with an additional buffer to store the prefetched data, as shown in Figures 2.5(a) and 2.5(b). Some recent commercial processors, however, use bufferless prefetchers, which directly prefetch the data into the last level cache (LLC). Compared with buffering, the bufferless prefetcher reduces on-chip storage cost. Bufferless options, however, may cause cache pollution when prefetching the wrong data or the right data at the wrong time.

Figure 2.5 shows three possible prefetching schemes for the HHT, namely (a) an off-chip prefetcher that prefetches data to an off-chip buffer; (b) an on-chip prefetcher that prefetches data to an on-chip buffer, and (c) a LLC prefetcher that prefetches data to LLC. In Section 2.5, we will quantitatively compare these prefetching schemes.

2.4 Methodology

We simulate our HHT systems on gem5 [12], a detailed, event-driven system simulator. We used the system emulation (SE) mode, as it is more efficient and it also supports simple address translation in the TLB, which helps simulate our addressing scheme. Section 2.4.1 explains the system simulation parameters in detail. Section 2.4.2 describes multiple benchmark kernels to drive our evaluation. Finally, Section 2.4.3 discusses several baseline systems that will be compared to our HHT systems.

2.4.1 Simulation Configuration

The parameters of the processor and memory system used in our experiments are listed in Table 1. The processor configuration is similar to the ARM Cortex A9 single core processor [13], which has a superscalar pipeline that forwards eight instructions per cycle to the decoder. Two levels of caches are used with this processor. As the HHT system uses PCM, we simulate a multi-level PCM and a single level PCM (for base line systems) on the nvmain [14] memory simulator, which supports modeling of realistic timing and queuing delay of multi-level PCM. For single-level PCM accesses, the memory latency includes the memory

Table 2.1: Simulation parameters, similar to ARM Cortex A9 single-core processor

CPU	2 GHz, single core, out-of-order, alpha ISA, 8-issue width, 64-byte cacheline size
L1 $	32 KB I/D-cache, 4-way, 2-cycle latency, 6 MSHR
L2 $	512 KB, 8-way, 12-cycle latency, 16 MSHR
Memory Controller	10 ns latency
Memory	4 GB; 1 channel, 1 rank and 4 banks; Row buffer size 1KB Multi-level PCM latency for level N: read latency: $(120 * 2^{(N-1)})$ns write latency: $(150 * N * 2^{(N-1)})$ns

controller latency, memory bus transmission latency, and PCM read latency. For the multi-level PCM, the simulator models exponential read latency as more bits are read out. All data levels are available after the highest level is read out. For write requests to the multi-level PCM, the simulator needs to read out all existing bits first, modify the bits to be updated, and write all the bits back to the PCM cells in the end. Table 2.1 lists the timing configuration in detail. For most of our experiments, we used four-level PCM for the multi-level PCM, but we will also discuss using more levels in Section 2.5.4.

2.4.2 Benchmarks

To focus the evaluation on the hash tables' performance in particular, we take hash table kernels from three benchmarks: Internet routing, Wordcount, and Dedup. The Internet routing benchmark takes a real trace of router traffic and builds a hash table for the mapping of IP addresses and their next hops. Wordcount is a program that counts the frequency of each unique word in a large text file. Dedup is a data stream compression algorithm from the PARSEC Suite [15]. It exploits a hash table to store and look up data blocks to find compression opportunities.

Table 2.2 shows the workload characterization of the benchmarks. We use similar scale

Table 2.2: Hashtable benchmark workload characterization

Benchmarks	Number of Hash Entries	Number of Lookups	Simulated Period (ns)
Internet Routing	50,000	2,050,000	14.5
Wordcount	44,998	2,000,000	6.7
Dedup	45,941	1,139,905	0.82

workloads for the hash tables in three benchmarks to simplify comparisons: the hash tables need to store around 45,000 entries in total and respond to around two million lookups (including searching and insertion). The simulated period is the longest gem5 simulated seconds for each benchmark with 1,024 buckets of chained hash tables on the single-level PCM.

We experiment with four hash table configurations: 8,192, 4,096, 2,048, and 1,024 buckets. Varying the number of buckets helps us characterize hash table behaviors in different situations and compare the scalability of different hash table systems. Figure 2.6 plots the bucket length distributions with four hash table configurations for the total workload of the Wordcount benchmark. With fewer buckets, the average number of entries in the bucket increases and the bucket size becomes more nonuniform. In this situation, it is more challenging to efficiently store the hash table and traverse all entries in each bucket at the same time. Furthermore, we take snapshots of the hash table bucket length distributions in the middle of each benchmark's execution. Figure 2.7 shows snapshots of the Wordcount benchmark with 1,024 buckets. These snapshots are taken in the middle of the program execution when 20%, 40%, 60%, and 80% of the total execution is finished respectively, to give a picture of the intermediate hash table structure when lookups happen (how many entries to traverse in each bucket). We'll show that HHT can get both good storage efficiency and good performance when the bucket length scales dynamically.

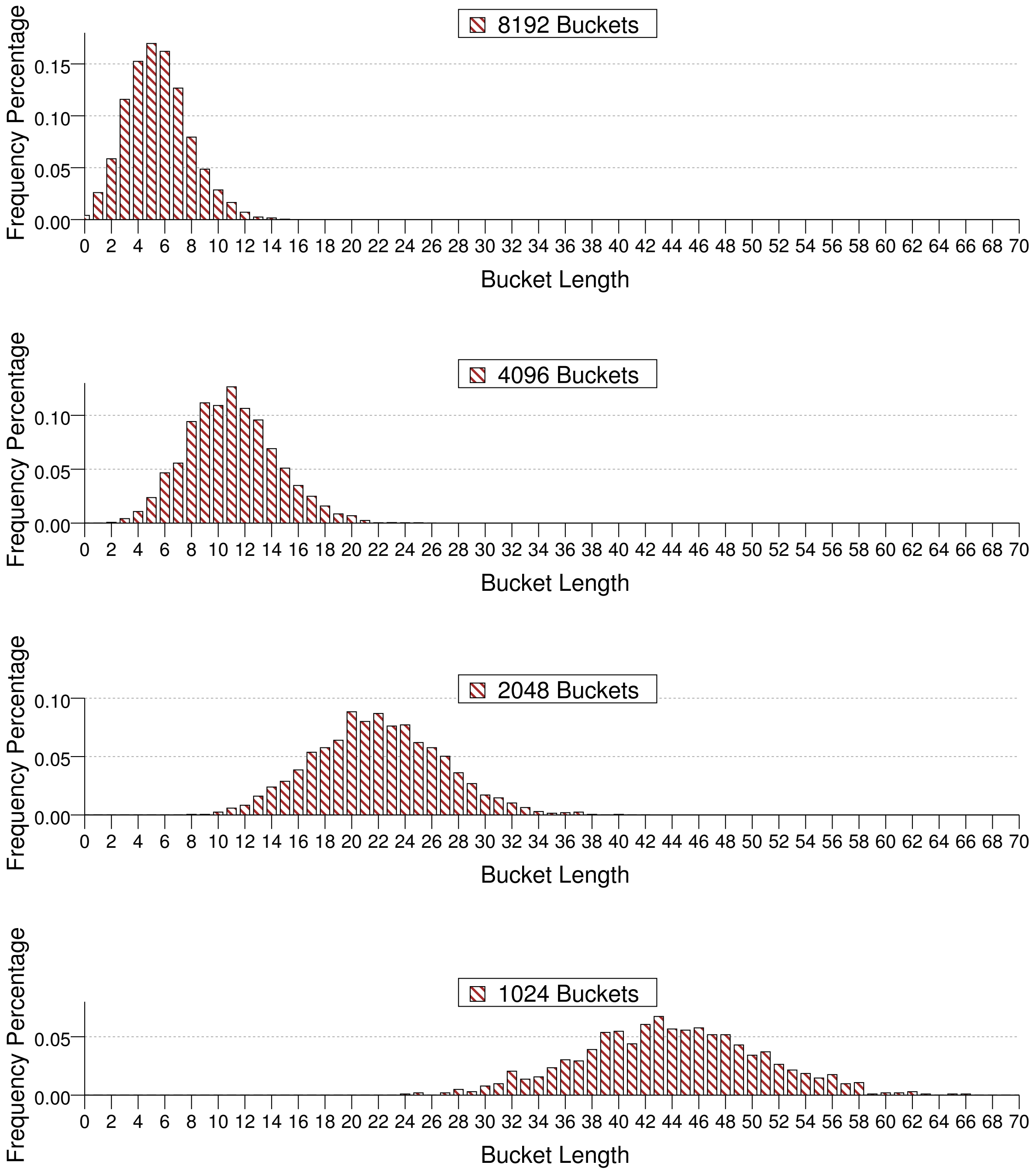


Figure 2.6: Wordcount using different numbers of buckets

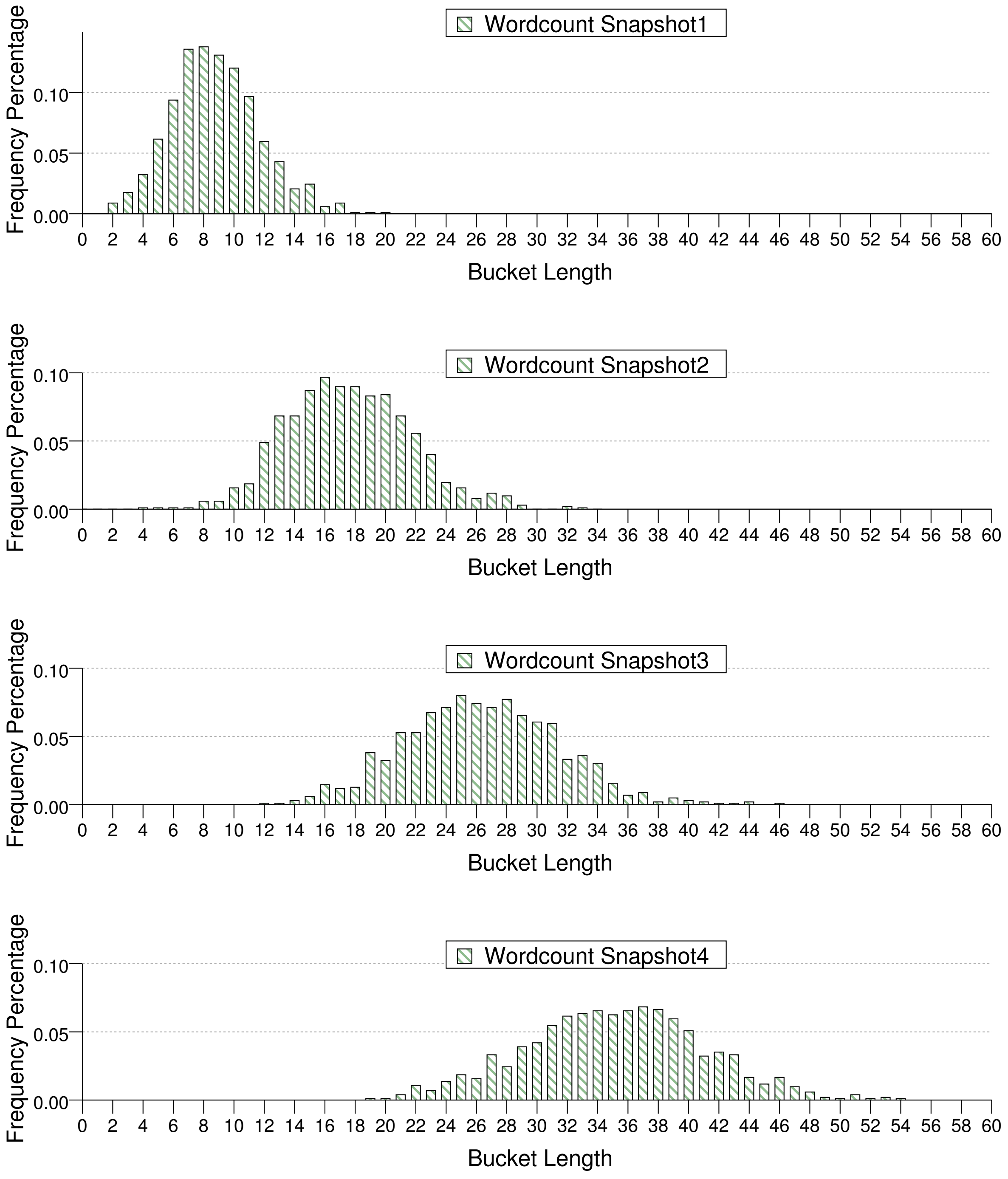


Figure 2.7: Snapshots of Wordcount with 1,024 buckets

Table 2.3: Hashtable system configuration

Baseline systems	Hash Node Structure	Memory Latency Model	Multi-level Storage
CHT on Single-level PCM	key, value, pointer to next	single-level	No
CHT on Multi-level PCM	key, value, pointer to next	multi-level	Yes
HHT without Prefetching	key, value	multi-level	Yes

2.4.3 Baseline Systems

We compare our HHT schemes with several baseline systems with different software and hardware implementations. We list the baseline systems and their difference in software implementation, hardware storage, and latency models in Table 2.3.

For chained hash tables (CHTs), we use a common implementation based on linked lists. The hash table maintains an array of linked lists, each of which stores hash entries that fall into the same bucket. Each linked list node contains three elements: the key, the value, and the pointer to the next entry in the same bucket. Chained hash tables on the single-level PCM is similar to the traditional hash table implementations on DRAM. The single-level PCM access latency is the sum of memory control latency, memory bus latency, and memory read latency.

To increase the storage efficiency of hash tables, we implement chained hash tables on the multi-level PCM as a baseline system in the following way: hash entries are dynamically allocated and randomly distributed in the multi-level PCM address space. In this system, the memory latency model follows the exponential read and write timing model as shown in Figure 2.2.

As discussed in the previous sections, the chained hash tables prevent both memory-level and instruction memory- level parallelisms. In the herniated hash tables (HHT) implementation, each entry contains just two elements: the key and the value. We use an array to maintain hash entries in the same bucket. There are no links between entries except when the number of collisions in one bucket exceeds the maximum level of the PCM. Taking an HHT implementation without prefetching as a baseline, we can see how much overhead is caused by the exponential latency model of the multi-level PCM and the effectiveness of prefetching schemes to tolerate this latency.

2.5 Results

In this section, we present evaluation results on HHTs, focusing on several issues.

First, storage efficiency analysis results are explained in Section 2.5.1, comparing HHTs with CHTs on single-level PCM and multi-level PCM, respectively.

Second, we present the performance analysis of HHTs in all benchmarks in terms of execution time and IPC (instructions per cycle). We also study the impact of the write cancellation technique and present the performance results with and without write cancellation.

Third, the impact of different prefetching schemes discussed in earlier sections is analyzed, in terms of execution time, LLC miss rate, average LLC miss latency and the memory traffic.

Finally, we explore the sensitivity of systems to using different numbers of buckets and deeper levels of PCM. All of the normalized metrics take HHTs without prefetching as the baseline.

2.5.1 Storage Efficiency Analysis

We define storage efficiency as the number of used memory cells here. Let N be the number of buckets in our hash tables and M be the number of unique keys in the workload. A HHT entry takes $16B$ and a CHT node takes $24B$. HHTs use the space of the array of head entries plus the space for redirected entries, which equals $(N + \textit{redirect pointers}) * 16B$. CHTs on the single-level PCM takes the array of head entry pointers plus the actual storage of all entries,

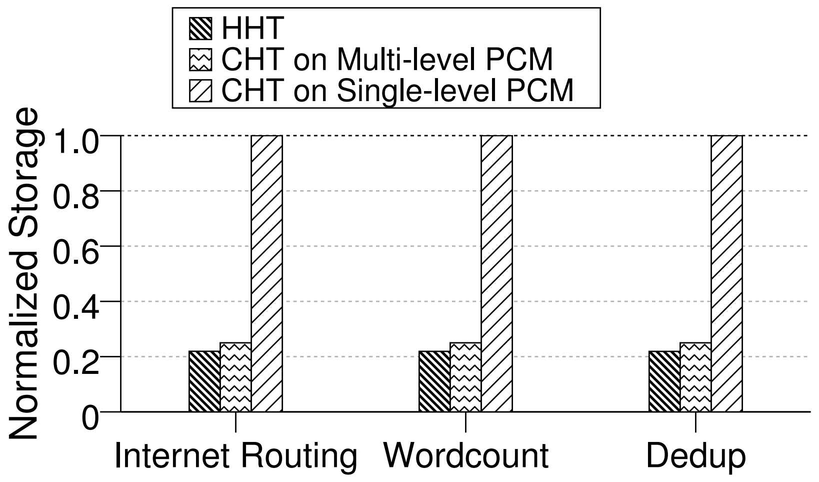


Figure 2.8: Storage efficiency of HHTs

which is $(N*8B+M*24B)$. Chained hash tables on the four-levels PCM has four times the address space of the single-level PCM, so it takes $1/4$ of the storage as on the single-level PCM.

Figure 2.8 shows the normalized storage efficiency results for HHTs and CHTs with 2048 buckets on the single-level PCM and on the multi-level PCM in three benchmarks. The HHTs get around 4.8x the benefits of the CHTs on the single-level PCM and 1.14x the benefits of CHTs on the multi-level PCM. When *M is much bigger than N*, HHTs have significant density benefits. We will discuss situations when *M is close to N* in the sensitivity study in Section 2.5.3.

2.5.2 Performance Analysis

Instructions per Cycle

For a given processor, the number of instructions per cycle (IPC) reflects the instruction-level parallelism that different systems can achieve. As discussed in the previous sections, herniated hash tables enable both instruction-level parallelism and instruction memory-level parallelism, so we can see substantial improvements in IPC.

Figure 2.9 plots the normalized IPC of three benchmarks with different hash table systems. We use 2048 buckets for all hash table systems to simplify the comparison. Performance with different number of buckets is shown in the sensitivity study in Section 2.5.3. The normalized IPC results in Figure 2.9 show that HHTs can achieve a 60% improvement over CHTs on the single-level PCM and about a 4x improvement over the CHTs on the multi-level PCM in terms of IPC. HHTs with LLC prefetching can achieve up to 1.66x improvement over HHTs without prefetching.

Execution Time

We also plot the execution time of benchmarks with different hash table systems to conduct a general performance evaluation. As shown in Figure 2.10, HHTs with LLC prefetching are up to 1.66x faster than HHTs without prefetching and more than 2.4x faster than the CHTs on the multi-level PCM. Compared with the CHTs on the single-level PCM, The HHTs with prefetching are 44% faster while achieving significant density benefits, as indicated in Section 2.5.1.

Write cancellation

Write cancellation [1] is an existing work to reduce read latency by canceling the already scheduled write requests if a read request arrives to the same bank within a period. As the write latency of multi-level PCM is very high, the write requests could significantly increase the effective latency of later arriving read requests. In most of our experiments, we use write cancellation as a built-in technique in multi-level PCM. In this section, we study the performance of our hash table systems if using multi-level PCM without write cancellation.

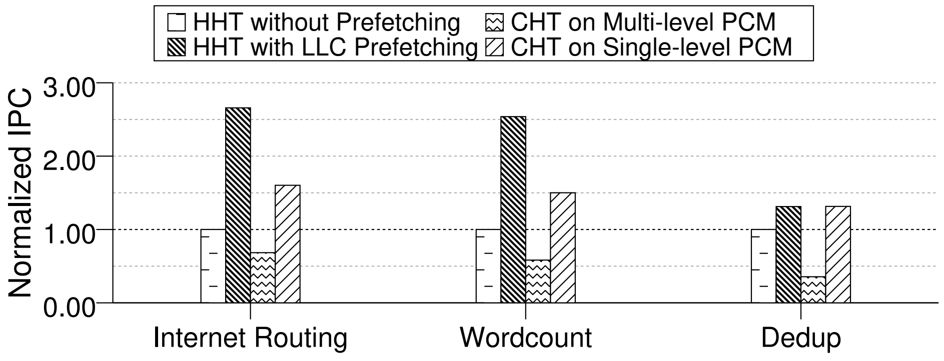


Figure 2.9: IPC of the benchmarks with different systems

Figure 2.11 and Figure 2.12 show the IPC and execution time results without write cancellation. For the Internet routing and Dedup benchmarks, IPC and execution time have no much difference using multi-level PCM with or without write cancellation because the read requests are much more than write requests. However, in the Wordcount benchmark, the number of write requests is similar to the number of read requests. The long write latency may cause a decrease in the system performance. From Figure 2.11, HHTs with LLC prefetching still achieve higher IPC than CHTs on single-level PCM but the improvement decreases from 60% to 52%. From Figure 2.12, the correspondent speedup of HHTs with LLC prefetching over CHTs on single-level PCM decreases from 44% to 27% for the Wordcount benchmark.

2.5.3 Sensitivity to Different Hash Table Configurations

Hash table behavior varies considerably when we configure them with different numbers of buckets, as shown in the workload characterization histograms. When fewer buckets are used, the number of collisions in each bucket increases. In this part of the experiments, we study the scalability of hash tables in both storage efficiency and performance in tolerating the increased number of collisions. We choose one representative benchmark, Wordcount, and study the

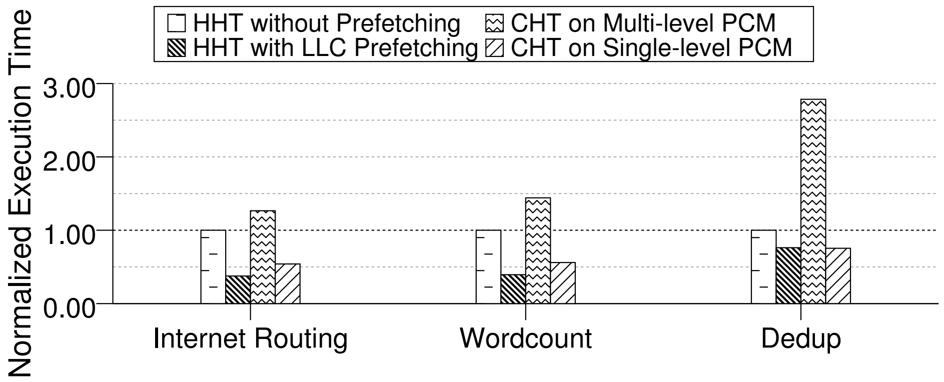


Figure 2.10: Execution time of the benchmarks with different systems

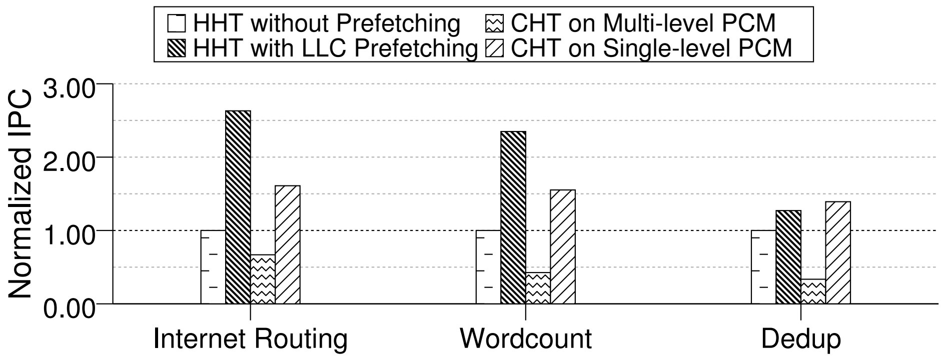


Figure 2.11: IPC of the benchmarks using multi-level PCM without write cancellation

hash table behavior with different numbers of buckets.

Storage

Varying the number of buckets can have the following effects on the storage. On the one hand, HHTs need more redirect pointers when fewer buckets are used while CHTs don't have this storage overhead. On the other hand, each hash table needs to keep an array of bucket head entries that is linear with respect to the number of buckets. In CHTs, this array stores just pointers (eight bytes) to the head entries. In the HHTs, this array stores real keys and values

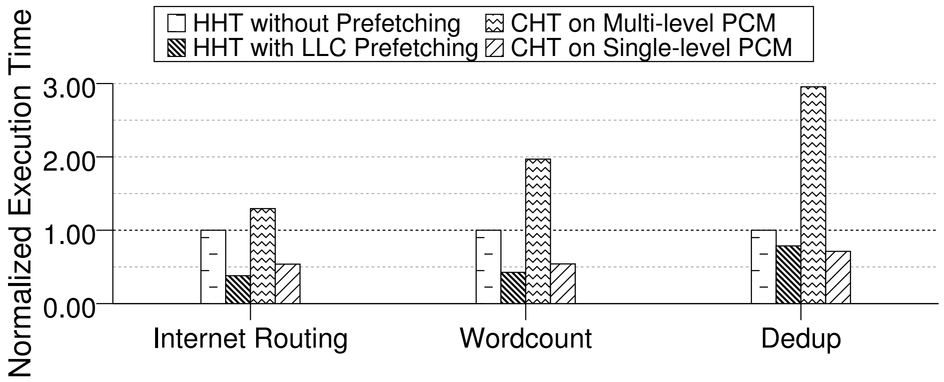


Figure 2.12: Execution time of the benchmarks using multi-level PCM without write cancellation

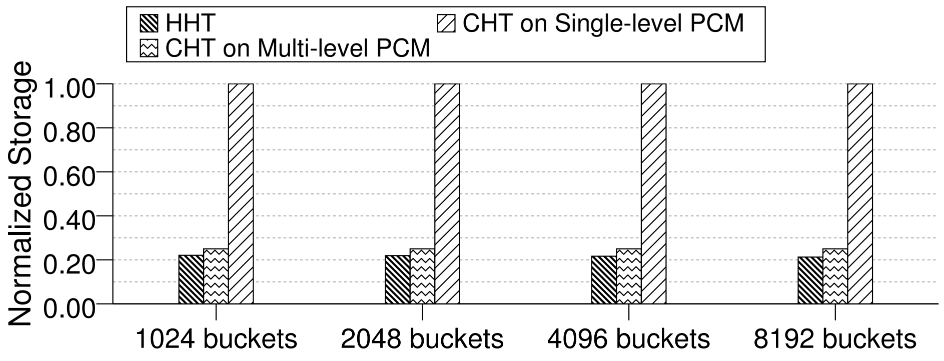
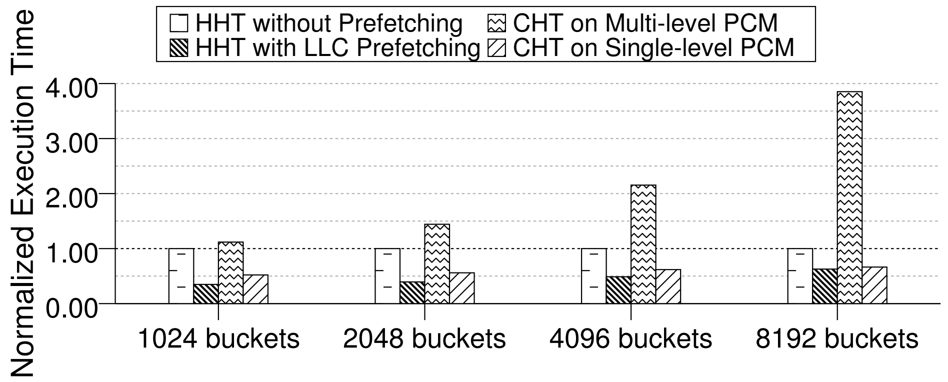


Figure 2.13: Storage efficiency of Wordcount with different numbers of buckets

(sixteen bytes). When the number of buckets is very large, there are many bucket head entries to store, even though some of the buckets are empty. HHTs have higher storage overhead for these head entries than CHTs. However, these two effects are in the opposite directions. Generally considering them together results in little difference in storage efficiency when the number of buckets changes, as shown in Figure 2.13. The HHTs are more storage efficient in all settings compared with CHTs on both the single-level PCM and the multi-level PCM.



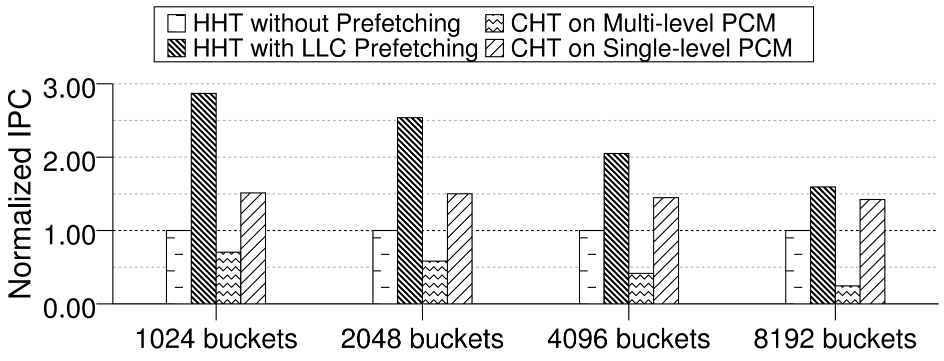


Figure 2.14: Performance of Wordcount with different numbers of buckets

Performance

Figure 2.14 shows the performance in terms of both execution time and IPC of Wordcount with different hash table systems and different numbers of buckets. For 8192 buckets, the HHTs with LLC prefetching are 12%, 59.4%, and 5.5x faster than CHTs on the single-level PCM, HHTs without prefetching, and CHTs on the multi-level PCM respectively. When the number of buckets decreases, the number of collisions starts to increase. the HHTs can tolerate more collisions because there are more prefetching opportunities. The CHTs cannot prefetch because of the dependencies between data accesses in the chained structure, so their performance degrades more quickly compared with the HHTs. For 1,024 buckets, HHTs with LLC prefetching show much better performance compared with CHTs on the single-level PCM by

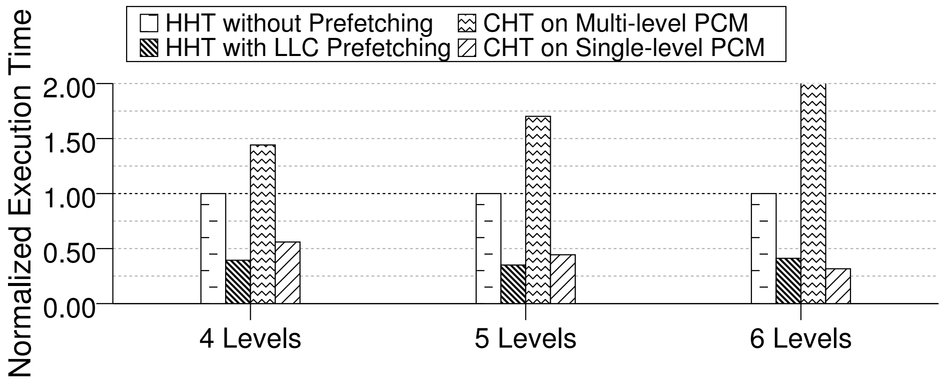


Figure 2.15: Sensitivity to PCM levels

achieving 1.87x IPC and 49% speedup in execution time.

2.5.4 Sensitivity to PCM Levels

When the PCM has more levels, herniated hash tables can yield more benefits by deeper prefetching and more overlap between computation and memory accesses. However, since latency is exponential with respect to the depth of the level accessed, there is a significant overhead when the highest level is accessed from PCM. In this section we study the performance of HHTs when using PCM with more levels.

Figure 2.15 shows the execution time of hash table systems using 4 levels, 5 levels, and 6 levels PCM respectively. Comparing the execution time of HHTs with and without prefetching with different levels, we can see that the contribution of the prefetching scheme increases by 12% from 4 levels to 5 levels because of more prefetching opportunities. However, the prefetching is less effective with 6 levels because the prefetching of one highest level entry will increase system latency significantly if the prefetched data is not used. In this situation, CHTs on the single-level PCM can get the best performance, but without density benefit. One future work is to study how to utilize different amount of levels in PCM dynamically for HHTs

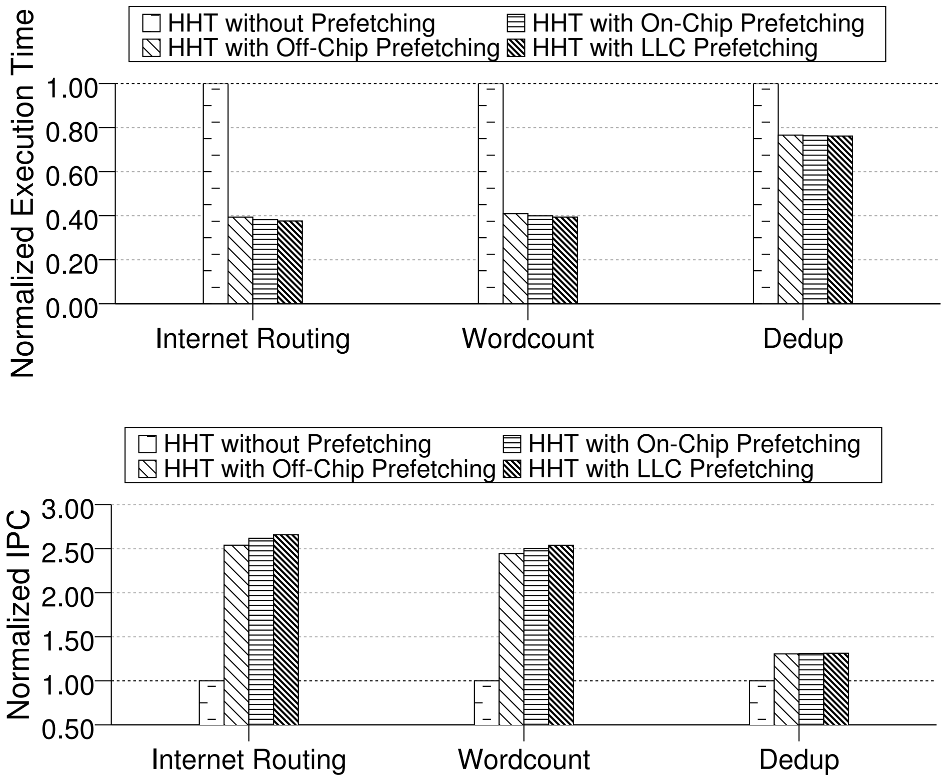


Figure 2.16: Performance of different prefetching schemes

to yield the best performance for different applications while getting the density benefit as well.

2.5.5 Impact of Different Prefetching Schemes

We discussed three possible prefetching schemes for herniated hash tables: off-chip prefetcher, on-chip prefetcher, and LLC prefetcher. These three options were motivated by the idea that we want to prefetch the data close to the processor to maximize the parallelism between instructions and memory. The off-chip prefetcher operates alongside the memory row buffer, which reads out all levels of data iteratively on a low-level access. This prefetcher saves the memory read latency in the multi-level PCM. The on-chip prefetcher prefetches all levels of data to an on-chip buffer in the memory controller. It saves the transmission latency on the

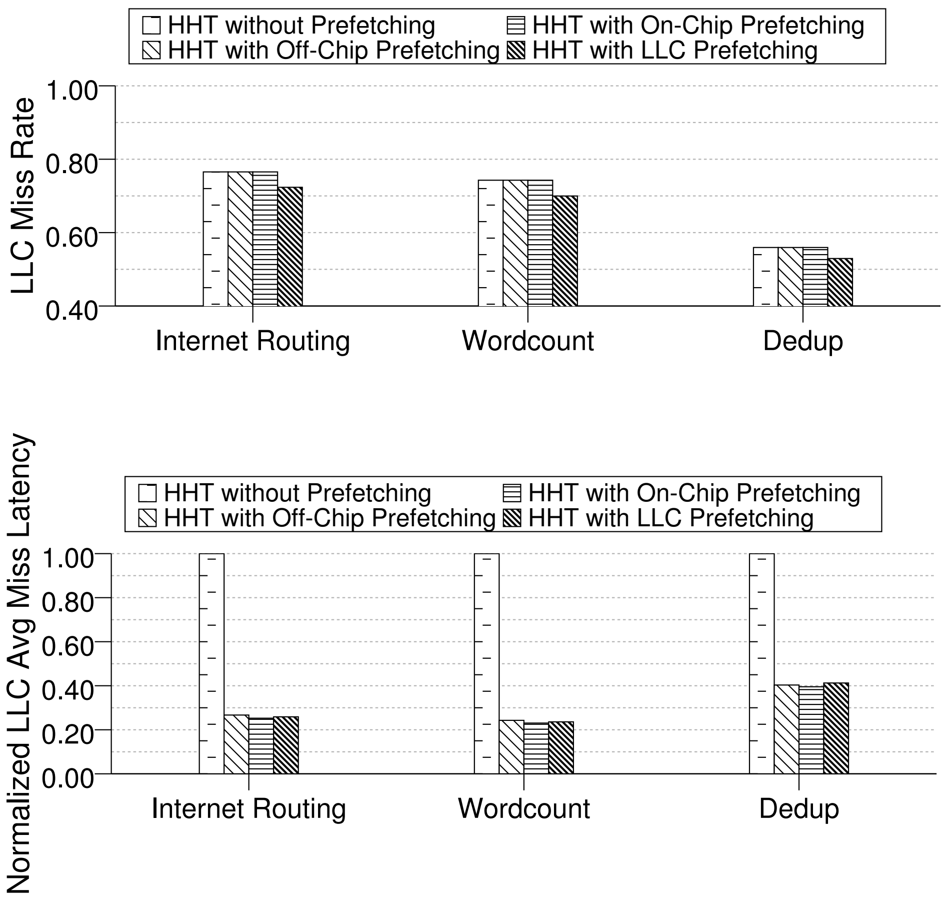


Figure 2.17: LLC Performance with different prefetching schemes

memory bus at the expense of a little more memory traffic. The LLC prefetcher moves the prefetched data into the last-level cache. It is the closest to the processor but may cause the cache pollution problem. Figure 2.16 shows the experimental results of the prefetchers, with 2,048-bucket hash tables.

We evaluate three benchmarks with all prefetching schemes. The HHTs without prefetching are taken as the baseline. From the execution time and IPC results as shown in Figure 2.16, we can see that the LLC prefetcher exhibits the best performance in general. The main reason is shown in the LLC miss rate result in Figure 2.17. With the LLC prefetcher, the LLC miss rate in the HHTs is reduced by about 6%, while the LLC average miss latency is similar to that of the other two prefetchers. Both the LLC prefetcher and the on-chip prefetcher need to fetch data to on chip storage. They achieve a little better performance than the off-chip at the expense of memory traffic. We quantify the memory traffic overhead of the on-chip prefetching, which is about 8.9% increase.

2.6 Related Work

2.6.1 Multi-level Phase Change Memories

Recently, a number of studies have discussed how to support multi-level PCM (and other multi-level non-volatile memory) on both hardware and software sides. Qureshi et al. [16] used a hardware-software hybrid scheme to improve the performance of multi-level PCM by converting the most often used multi-level pages into single-level PCM pages. Zhou et al. [17] extended this to a scheme for the virtual machine environment. Jiang et al. [18] focused on improving the performance of multi-level PCM writes with the assistance of ECC bits in memory. Joshi et al. [19] proposed controlling the write pulses of multi-level PCM, thereby improving its performance as well as reducing the energy consumption. Saadeldeen et al. [20] proposed using memristors to construct a branch predictor. Zhang et al. [3] proposed a sparse directory architecture utilizing multi-level memristors (a resistive memory technique similar to PCM). Sampson et al. [21] used an approximate technique to improve the performance, density, or lifetime of multi-level PCM. Our work is unique in that it exploits nonuniform density to adapt to asymmetric datastructure growth and uses a physical addressing scheme that integrates well with the cache hierarchy.

2.6.2 Hashtable Acceleration

A great deal of work has been done to optimize the performance, storage, and scalability of hash tables. The use of concurrent hash tables [22] is popular which are featured with lock-free extensive hash tables enabled by techniques such as “recursive split ordering” [23]. There has been other work to utilize techniques such as the read-copy update mechanism [24] or software transactional memory [25] to achieve better performance, scalability, or both. Additionally, there has also been work focusing on novel hash tables, such cuckoo hash [26], for multiple concurrent writers and readers.

2.7 Chapter Summary

Datastructure accelerations are of great importance in the big data era. Hashtables are a widely-used datastructure due to their average constant complexity for accesses. However, when more and more collisions occur, the traditional chaining implementation prevents two kinds of parallelism: memory-level parallelism and instruction-memory level parallelism. Multi-level PCM is an emerging memory technique that may be an alternative of DRAM, as it brings significant density benefits by supporting multiple resistance levels at each cell. In this chapter, we introduced Herniated Hash Tables, which utilizes the multi-level PCM to store multiple hash collisions in the same bucket at the same PCM cells. However, the density benefits of multilevel PCM come with the exponential latency overhead when reading more than 1 bit per cell. To avoid this latency, we propose three prefetching options: Off-Chip prefetcher, On-Chip prefetcher and LLC prefetcher. Experimental results show that Herniated Hashtables can get up to 4.8x density benefits while achieving up 67% performance speedup over traditional chained hash tables on single-level PCM.

Chapter 3

Lemonade from Lemons: Harnessing Device Wearout to Create Limited-Use Security Architectures

3.1 Introduction

Wearout is not a new problem in computer architecture. Flash memories are the most well-known memory technologies with the wearout problem, and people have been working on efficient solutions to it for decades. In the big data era, the problem is even worse with continuously increasing density and capacity demand for memory. The lifetime of a flash cell dropped from 10,000 times to 2,000 times when the cell dimension scales down from 50nm to 20nm [27]. Moreover, ITRS [28] envisions many new technologies such as non-volatile memories (NVM), nanoelectromechanics (NEMS), and molecular devices, to solve the power-consumption problem in CMOS so as to sustain Moore's law. However, the wearout problem also exists in these technologies [29, 30, 31, 32]. The down-scaling from MEMS to NEMS, for instance, results in exponential degradation of device reliability [33, 34]. Representative NEMS contact switches can only work for one cycle to several thousand cycles without failures [33, 35, 36, 37]. As a result, existing research on these emerging technologies has been focused on how to extend the lifetime before they can really be applied.

Meanwhile, we take a step back and find that wearout can help build purposely limited-use security architectures. In applications that abusive accesses or adversarial accesses are not desired, wearout can provide strong security by automatically destroying the device and hence protects any secure information in it. For example, forward secrecy encryption [38, 39] in any public key systems [40, 41] (eg. the encryption of e-mail archives) requires a one-time key for the encryption of each message so that the compromise of a single private key does not compromise all the past messages. Traditionally, the one-time access of the keys is not enforced so the system still cannot defend against reusing or stealthy replications of the keys. Taking advantage of wearout, we can store the keys in a security architecture that wears out exactly after one access so that the one-time usage of keys is physically enforced and the security of messages will not be compromised.

However, taking advantage of wearout to create hardware security mechanisms is challenging. The problems we face in designing the security architectures include:

- How do we design for system-level minimum and maximum usage in the face of probabilistic wearout behaviors of each device and process variations among devices?
- Depending on security goals and usage targets, how should we adjust the parameters in our design to minimize area and energy cost?
- How do we balance the fabrication cost of more consistent devices (in terms of wearout) with the area cost of architectural techniques to achieve consistency (eg. redundancy and encoding)?

In this work, we propose a methodology to create security architectures by harnessing device wearout. Our contributions are as follows:

- We use the two-parameter Weibull distribution to model the time to failure of each device. The two parameters in the model can be used to characterize different kinds of wearout devices. Manufacturing and process variations among individual devices are accommodated by introducing more variations into the distribution.
- Based on the probabilistic wearout model, we provide statistical guarantees on system-level usage bounds by designing application-dependent architectures and exploiting redundant encoding techniques. The system-level usage bounds ensure both reliability for legitimate users and security to defend against brute-force attackers.
- Extensive engineering space exploration has been performed to study the trade-offs among target access bounds, area cost, fabrication cost, etc.

In the following sections, we first introduce the wearout devices used in our security architectures and describe the probabilistic wearout model in Section 3.2. Then we talk about the threat model in Section 3.3 for three use cases of hardware security mechanisms: a limited-use connection, a limited-use targeting system and one-time pads, discussed in Section 3.4, 3.5 and 3.6, respectively. Note also, we talk about the limitations of the degradation-based security measures in Section 3.7. Then we discuss the literature of hardware security in Section 3.8. Finally, we conclude our work in Section 3.9.

3.2 Device wearout model

3.2.1 NEMS contact switches

NEMS contact switches exhibit promising properties such as nano-scale physical dimensions, near zero OFF-state leakage, and large ON/OFF ratios. Representative NEMS contact switches are composed of a movable active element and an opposing electrode which interact by both electrical and mechanical forces to open and close the switch. When a pull-in voltage is applied to the active element, electrostatic forces deform the active element towards the opposing electrode so as to close the switch. After the voltage is removed, elastic forces pull the active element away from the opposing electrode so that the switch is open again.

Among other technologies, NEMS are advantageous in building limited-use security architectures because they have relatively tight wearout bounds and they are insensitive to harsh environments [33] including radiation, temperature, external electric fields, etc. Most fabricated NEMS switches can work properly for only one to several thousand cycles [33, 35, 36, 37]. Recently, NEMS lifetime can be extended up to millions or billions of cycles while at the cost of scaling up the physical dimensions [42, 43, 44]. The wearout characteristics of NEMS switches are highly dependent on the materials and structures employed. Generally speaking, any kind of electrical and mechanical aging, adhesion, fracture or burnout in the active element or electrode are potentially responsible for the failures of NEMS switches. For example, in [35], graphene-based NEMS switches were recorded to work over 500 cycles and then failed because the over-bent graphene was unable to recover. [36] reported that NEMS switches with silicon carbide nanowires can switch for tens of cycles before the nanowire was stuck to the electrode. In [45], the silicon carbide cantilevered NEMS switches failed after billions of cycles because of fracture at room temperature and melting at 500 °C.

Furthermore, due to their insensitivity to harsh environment, NEMS switches can help defend against attacks by varying the dynamic environment. For example, it is hard for attackers to extend the lifetime of NEMS switches by controlling the operating temperatures, especially these made of high temperature friendly materials such as SiC. Poly-SiC NEMS switches [42] were recorded to operate properly for at least $10^5 \sim 10^6$ cycles without failures at 500 °C , comparable to those at 25 °C. [45] has shown that SiC NEMS switches can operate more than 21 billion cycles at 25 °C while more than 2 billion cycles at 500 °C. However, failures at 25 °C were characterized by fracture while failures at 500 °C were probably caused by melting.

For security architectures, we assume the lifetime at room temperature (25 °C) as the device wearout bound. More failures at extremely high temperatures will destroy the device faster, but will not compromise the secret information in them. At extremely low temperatures (eg. after freezing), it is hard to extend the device lifetime either because failures from fracture cannot be avoided. These features have enabled NEMS switches to be applied to security or harsh environment applications, eg. one-time-programmable FPGA interconnects in [46].

3.2.2 Probabilistic wearout model

In the reliability literature [47, 48], Weibull distributions have been commonly used to model the failure distributions of electronic devices [49], such as the breakdown of gate oxides in CMOS [50]. Similar models can also be applied to micro-/nano- scale devices [51]. It has been shown that the Weibull distribution can accurately fit fracture-strength data of emerging materials for NEMS/MEMS [52, 53], fracture-test data of cantilever beam MEMS [54], and tensile-strength data sets of carbon nanotubes [55].

Hence, we take the general two-parameter Weibull distribution to model the failure distribution of NEMS switches in this work. Assume that x represents the time to failure. Then the probability density function (PDF) of the time to failure is:

$$f(x) = \frac{\beta}{\alpha}\left(\frac{x}{\alpha}\right)^{\beta-1} e^{-\left(\frac{x}{\alpha}\right)^{\beta}} \tag{3.1}$$

and the cumulative density function (CDF) is:

$$F(x) = 1 - e^{-\left(\frac{x}{\alpha}\right)^{\beta}} \tag{3.2}$$

Based on the CDF, the reliabity function is derived as follows:

$$R(x) = 1 - F(x) = e^{-\left(\frac{x}{\alpha}\right)^{\beta}} \tag{3.3}$$

In all of the above equations, α is the scale parameter and β is the shape parameter. The two parameters can be estimated by fitting the lifetime data of a large population of similar devices. α approximates to the mean time to failure, and β mainly determines the variation of reliability

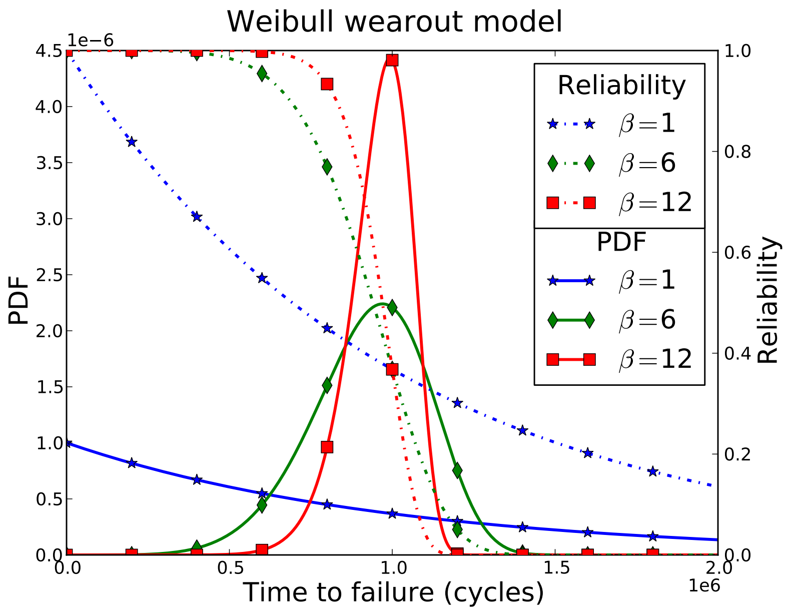


Figure 3.1: Weibull wearout model with different shape parameters. The red lines ($\beta = 12$) show the lifetime plots of MEMS devices [56] with geometrical variations.

degradation among these devices.

Figure 3.1 shows the failure PDF (Equation 3.1) and reliability function (Equation 3.3) with different βs. The variation of α will only change the axis scales. β is usually larger (sharper peaks in the PDF) with more homogeneous devices in which the wearout happens more consistently.

Since the manufacturing processes are less mature at nano-scales, we accommodate the process variations in αs and βs. Typically, process variations will result in lower βs. Trevor S. Slack et al. [56] has simulated Weibull lifetime models of MEMS devices considering geometrical variations, material variations in elastic modulus, resistance stress, etc. According to their simulation results, αs and βs are 2.6 million cycles and 12.94 with only geometrical variations, 2.2 million cycles and 7.2 with material elasticity variations, 1.8 million cycles and 8.58 with material resistance variations. We will experiment with various αs and βs for an extensive engineering space exploration in later sections.

3.3 Threat model

Since our target devices are often mobile devices, we assume the attacker has physical access to the device. In the cases of the limited-use connection and targeting system, we want to defend against brute-force attacks to decrypt the device by physically limiting the number of accesses to the storage decryption key. The cracking approaches we target are professional attacks that can exploit the nonuniform guessability in real-world passwords [57], as discussed in Section 3.4.1.

In the case of one-time pads, we want to defend against stealthy replications of device (the archaically named "evil maid attack"). The attackers may clone the one-time pads and make two copies, one copy to replace the receiver's original one and another copy for themselves to break the encryption in the future message transmission between the sender and receiver. Our secure architectures will resist cloning by making it difficult for attackers to ever read the entire contents of the device memory.

In this work, we assume the chip fabrication is trusted. And we leave as future work techniques to allow secure, one-time programming of our devices by end users. We assume the secret information is one-time programmed in the device memory at fabrication time and end-users will only need read operations through the NEMS network.

3.4 A Limited-use connection

Apple iOS has developed many security features with integrated secure software and hardware support. However, to keep the devices easy to use, the most straightforward way to protect the devices is to use a passcode that is configurable by users. To protect the passcode from brute-force attacks, iOS provides several mechanisms [58]: 1) it automatically wipes out all data on the device if someone has consecutively failed 10 times in unlocking the device. 2) it has incremental time delays after each incorrect passcode attempt.

However, all these guarding mechanisms are recently reported to be easily bypassed. A company called MDSec [59] managed to bypass the internal counter in iPhone by cutting its power right before the counter is incremented, while still getting the passcode validation result. Therefore, the counter never gets updated and the attacker can break the passcode as fast as the hardware can support. [60] demonstrated similar attack to the counter through NAND mirroring. An iPhone 5c was forced to power down and recover from a backup NAND memory to restore the previous state once every a few passcode attempts, which enabled unlimited attempts to crack the passcode. Another hacking case exploited firmware updates [61]. iPhone can launch firmware updates automatically without the passcode, which means that the guarding mechanisms are disabled or the counter is disabled during the updates. Then brute-force attacks can easily succeed, especially with real-world biased passcodes.

Although some vulnerabilities mentioned above have been patched, the real problem with software solutions is that they can not prevent unknown vulnerabilities. Therefore, we propose to exploit NEMS switches to build a limited-use connection that can physically limit the number of accesses to the storage decryption key. The right passcode and the storage decryption key are needed to successfully decrypt the device storage and thus validate the passcode. We enforce a traversal to the limited-use connection before each read of the storage encryption key. After each passcode attempt, the failure probability of the connection increments. After a certain number of accesses, the connection will wear out automatically and the smartphone will be locked forever. Compared with security patches, our solution provides strong hardware enforced security, which cannot be compromised even under government's intervention.

In Section 3.4.1, we talk about the design principles and design options for the limited-use connection using NEMS switches. And we discuss system integration issues with NEMS switches in Section 3.4.2. Finally, we explore the engineering space given the design options in Section 3.4.3.

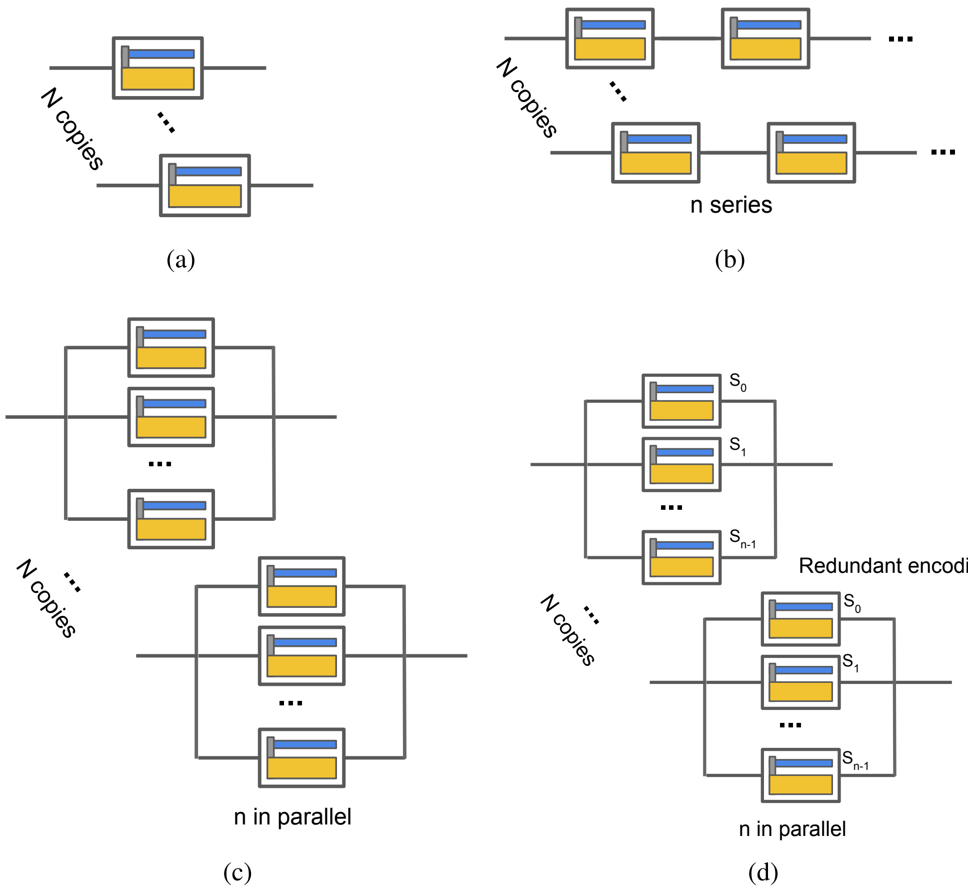


Figure 3.2: Design options for the limited-use connection using NEMS switches. Fig 3.2a uses N copies of single NEMS switches. Fig 3.2b uses N copies of n NEMS switches in series. Fig 3.2c uses N copies of n NEMS switches in parallel. Fig 3.2d uses same parallel structures but with redundant encoding.

3.4.1 Using wearout to build a limited-use connection

The design of the limited-use connection needs to follow two principles:

- The connection should work reliably for all legitimate accesses during the smartphone's lifetime.
- The connection should wear out before attackers have high probabilities to guess the passcode.

As an example, we calculated the legitimate access bound (LAB) for a smartphone approximately as follows:

$$LAB = 5 * 365 * 50 = 91,250 \tag{3.4}$$

If the real LAB is several times larger than that, we provide M-way replication of our entire architecture to scale the LAB by a factor of M, as described in Section 3.4.1.

The main challenge in designing the security architectures is how to control the system-level reliability degradation window $[t_1, t_2]$, as indicated in the two principles. The lower bound t_1 should be greater than the LAB while the upper bound t_2 should be less than the minimum guesses needed to crack any passwords. According to Blase Ur et al.'s work of measuring real-world password guessability [57], professional attackers usually try passwords in the order of empirical popularity. For 8-character passwords including characters from all different classes (lowercase letters, uppercase letters, numbers, and special characters), only a few very popular passwords can be guessed within 91,250 attempts. The guessing probability increases to 1% and 2% with 100,000 and 200,000 attempts, respectively.

In the following sections, we first attempt to design security architectures that wear out as quickly as possible after the LAB. We focus on seeking for architectural techniques that can help control the degradation window. Then we extend the upper bound to 100,000 and 200,000 accesses if the software helps reject the most popular 1% and 2% passwords, discussed in Section 3.4.3.

According to the design principles, we consider four possible design options, as illustrated in Figure 3.2. These design options are explained in detail as follows.

N copies of single NEMS switches

With a single NEMS switch, it is difficult to meet the system-level demand of minimum and maximum accesses. On one hand, the empirical lifetime of a single NEMS switch is usually not as long as the LAB. On the other hand, even if there exists such a NEMS switch, the degradation window expands millions of cycles, as shown in Figure 3.1. Thus, we consider using N copies of NEMS switches, which can divide the system-level access bounds by a factor of N. Then the design principles for each copy are adjusted as follows:

- Each copy should work reliably for $\frac{\text{LAB}}{N}$ accesses.
- Each copy should wear out before $\frac{\text{LAB}}{N}+1$ accesses.

Although the second requirement still requires small degradation windows, the first requirement scales the mean value down to $\frac{\text{LAB}}{N}$. This down-scaling helps create a small degradation window, as shown in Figure 3.3a.

However, when α is very small, the lifetime of the device is expected to be very small. The manufacturing and process variability is probably hard to control for such brittle and fragile devices. We will discuss more architectural options next to avoid such challenges in fabrication while still satisfying both design requirements.

N copies of n NEMS switches in series

Instead of looking for NEMS switches that can fail extremely fast, we consider chaining NEMS switches in series to accelerate the wearout, as in Figure 3.2b. In the chaining architecture, if any single NEMS switch in the chain fails, the whole chain fails. Unfortunately, we found that increasing the number of NEMS switches in the chain has no significant impact on the failure rate.

Assume we have n NEMS switches in series. Then the reliability of the chain is:

$$R(x)=\left(e^{-\left(\frac{x}{\alpha}\right)^{\beta}}\right)^{n}=e^{-n\left(\frac{x}{\alpha}\right)^{\beta}} \tag{3.5}$$

Compared with the reliability function for a single device in Equation 3.3, n devices with α in series are equivalent to a single device with $\frac{\alpha}{n^{(1/\beta)}}$. Let the denominator equals y: $y=n^{(1/\beta)}$. Then we get: $n=y^{\beta}$. If we want to increase y to scale down α as in Figure 3.3a, then n

should increase to y^{12} in each copy with $\beta = 12$. To avoid the explosion of NEMS switches, we discard this option in the following discussion.

N copies of NEMS switches in parallel

Here we propose another technique to control the degradation window, which is exploiting parallel structures to improve the system reliability before all devices wear out, as shown in Figure 3.2c. Assuming N copies of parallel structures, the requirements for each copy are as follows:

- At least one NEMS switch in each copy should work reliably for $\frac{\text{LAB}}{N}$ accesses.
- All NEMS switches in each copy should wear out before $\frac{\text{LAB}}{N} + 1$ accesses.

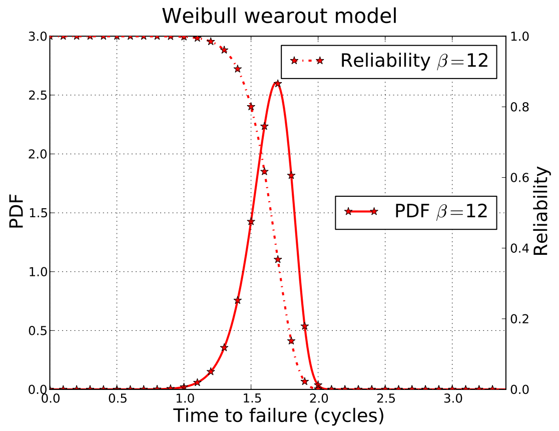
Assume each copy has n NEMS switches in parallel. The reliability of this parallel structure is :

$$R(x) = 1 - \left(1 - e^{-\left(\frac{x}{\alpha}\right)^{\beta}}\right)^{n} \tag{3.6}$$

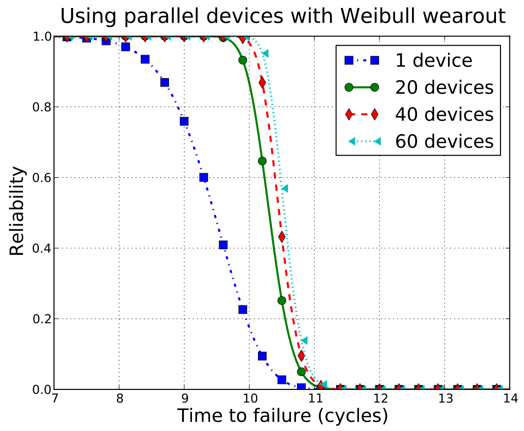
The redundancy in the parallel structure provides error tolerance so that the high reliability threshold is pushed toward the degradation edge, as shown in Figure 3.3b. With 98% reliability, the parallel structure with 40 devices can work for the 10th access. With only 2.2% probability the parallel structure will continue working for the 11th access. In this design option, the total number of devices may increase, from 91,250 to 365,000 ($40 * (91,250/10) = 365,000$) approximately, but the mean time to failure of NEMS switches is relaxed from one to about ten cycles.

Parallel NEMS switches with redundant encoding

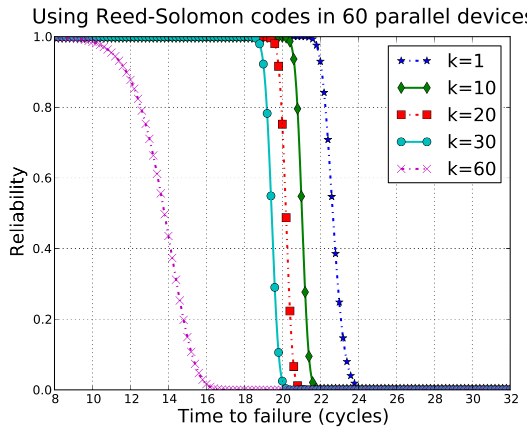
In this section, we introduce Shamir's secret sharing mechanism and redundant encoding techniques to further speed up the degradation of the limited-use connection. Instead of using highly redundant and reliable 1-out-of-n parallel structures, we require at least k NEMS



(a) By scaling α down, we get a degradation window within 1. Given $\alpha = 1.7$ and $\beta = 12$, the reliability of each device is close to 1 when $t = 1$, but close to 0 when $t = 2$.



(b) Using parallel NEMS switches to push the high reliability threshold toward the degradation edge. $\alpha = 9.3$ and $\beta = 12$ for all NEMS switches.



(c) Using redundant encoding in parallel structures to accelerate degradation. $\alpha = 20$ and $\beta = 12$ for all NEMS switches.

Figure 3.3: Different techniques to control the hardware degradation window

switches working in a parallel structure in order to decrypt the device storage, which results in a k-out-of-n parallel structure. Increasing k to some extent tightens the wearout bounds of each parallel structure so that the connection wears out faster. The challenging part, however, is that the k-out-of-n parallel structure should provide reliable connection when k or more NEMS switches work properly but wear out quickly when only $k-1$ or less NEMS switches do the same.

To enforce that, we exploit Shamir's secret sharing mechanism and redundant encoding techniques. The general idea is the following: We encode the storage decryption key into n components and spread them in n read-destructive storage connected by the NEMS switches in a parallel structure. The encoding mechanism enforces that at least k components are needed to successfully get the key while no knowledge about the key will be revealed with less than k components.

Shamir's secret-sharing scheme

Shamir's secret-sharing scheme [62] constructs fast degradation codes. One of its classical scenarios is the secret sharing among many people. On one hand, the scheme allows efficient and reliable secret sharing if at least k out of n people authorize accesses to the secret. On the other hand, the scheme prevents leaking any information about the secret if there is only authorization from $k-1$ people or less. The scheme is also called a (k,n) threshold scheme and its reliability degrades immediately at $k-1$. Unlike classical secret-sharing scenarios that tolerate partial errors for more efficient sharing (authorizing access with the majority of people's permission [62] or matching interests with the majority of attributes [63]), our security architectures need to tolerate erasures from device failures.

The encoding and decoding in Shamir's secret sharing scheme is based on polynomial construction and interpolation. The insight is that given k independent points on a 2D plane, there is one and only one polynomial of degree $k-1$ that passes through all the k points. The encoding algorithm involves constructing such a polynomial of degree $k-1$ with the secret hidden in the coefficients:

$$q(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \tag{3.7}$$

With this polynomial, we encode the secret into n points, for instance, $q(1)$, ..., $q(n)$. The n points are then distributed to n people (or n devices) in the security application. Given any k points, all coefficients can be easily computed by interpolation so as to recover the secret. With

$k-1$ points or less, however, no information about the secret can be inferred.

Error correction codes with fast degradation

Reed-Solomon codes [64, 63] are the error correction version of Shamir's secret-sharing scheme and they are commonly used in the error correction of large amounts of data in devices such as flash disks, CDs and DVDs. Theoretically, other linear codes could also construct similar (k,n) threshold secret sharing schemes, but it is hard to reason about the security because of the hardness of approximating the minimum distance of any linear code [65].

Figure 3.3c shows the reliability of a parallel structure with Reed-solomon codes. With redundant encoding, the reliability of this security architecture becomes:

$$R(x) = \sum_{i=k}^{n} \binom{n}{i} \left(e^{-\left(\frac{x}{\alpha}\right)^{\beta}}\right)^{i} \left(1-e^{-\left(\frac{x}{\alpha}\right)^{\beta}}\right)^{n-i} \tag{3.8}$$

We use 60 NEMS switches in the parallel structure and can relax the wearout bound for each NEMS switch to around 20 cycles. With $k=1$, the degradation window size is about 2, while with $k=30$, the degradation window size reduces to about 1, as shown in Figure 3.3c. With $k=30$, the parallel structure provides 92% reliability for the 20th access while only 2% probability for the 21st access. However, when k is close to the total number of parallel devices in the structure, the reliability degrades very early that the degradation window is stretched out again.

Assisted with Reed-Solomon codes, we are able to build the limited-use connection with approximately the same number of devices $(60*(91,250/20)=365,000)$ compared with the parallel structure without encoding, but we can further relax the wearout bound to 20 cycles. The overhead, however, includes the encoding/decoding complexity and extra storage for the component keys.

M-Way Replication of Modules

A legitimate usage factor of 50 times per day is potentially low for some users, so we propose to support increasing this usage by a factor of M by replicating our entire structure M times. The replicated modules must be used serially and each must employ a new password. In this way, an attacker can only attack each module separately to its upper access bound, but the user can achieve usage that is the sum of the lower access bounds of all M modules.

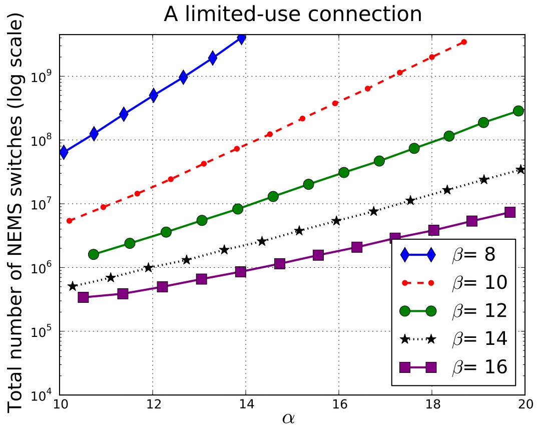
The cost of this scheme, however, is that a new password must be chosen when migrating from one module to another and the storage encryption key must be re-encrypted with the new password. For example, if we wish to increase usage from 50 times to 500 times per day, we use a 10-way replication factor, which implies that the user must choose a new password and re-encrypt storage every 6 months during our target 5-year lifetime of the smartphone.

3.4.2 System integration

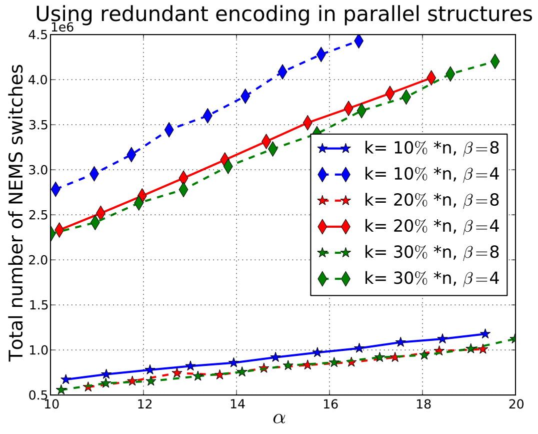
To build the limited-use connection in practical mobile devices, we need to integrate the NEMS network with conventional CMOS. Here we show that the CMOS-NEMS integration is feasible in the manufacturing process and the CMOS-NEMS interface does not compromise the security of the device storage.

Manufacturability NEMS devices are CMOS-compatible: they do not require exotic materials or fabrication flow. In the literature, there have been integrated CMOS-NEMS circuits for leakage-control and power-gating [66, 67]. A possible integration solution, for example, is to have NEMS and CMOS circuits in differet layers of the chip with a sandwiched metal layer in between [68].

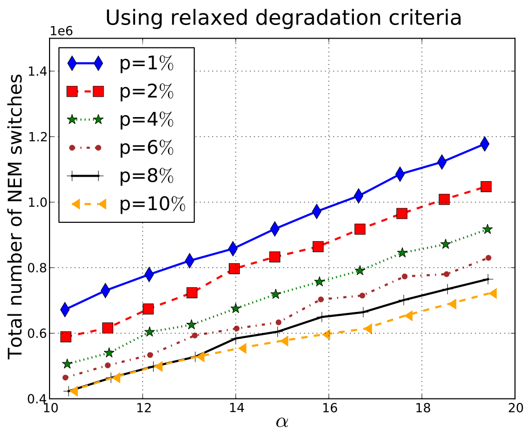
Security We bury the secret key many layers below the surface of the chip and only connect that secret key to the surface through NEMS devices. Although there are CMOS-NEMS



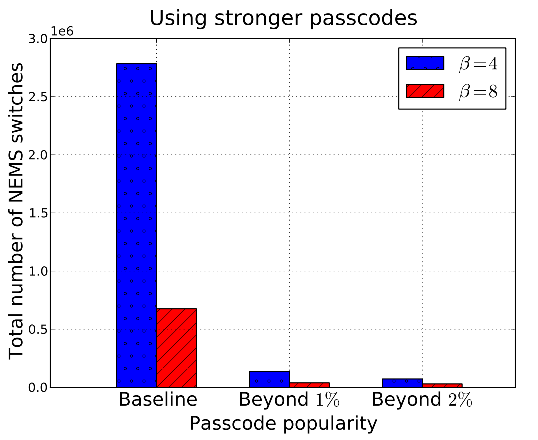
(a) Without redundant encoding



(b) With redundant encoding



(c) Relaxed degradation criteria



(d) Using stronger passcodes

Figure 3.4: The total number of NEMS switches needed with different engineering options for the limited-use connection.

connections both from the surface to the NEMS network and from the NEMS network to the deeply buried secret key, the latter connections are difficult to access and thus provide a level of physical security. Circumventing the surface connections does not help the adversary access the secret.

3.4.3 Engineering space exploration

In this section, we talk about the engineering space exploration for the limited-use connection. Without loss of generality, the discussion will be focused on architectural options using N copies of parallel structures with or without encoding. As discussed earlier, our goal is to guarantee the system-level access bounds to protect the passcode from real-world brute-force attacks. Here we list several parameters in the engineering space of the limited-use connection: device wearout characteristics, redundant encoding, and system-level access bounds. Next, we will discuss these parameters and their trade-offs in fabrication cost, area cost, encoding complexity, etc. The experiments are based on numerical simulations with different engineering options.

Device wearout characteristics

We first choose different αs and βs to characterize various kinds of NEMS switches. Since α approximates to the average lifetime of devices in the Weibull model, we set α according to the lifetime of representative NEMS switches as listed in [33], ranging from 1 cycle to 1000 cycles. In the following discussion, we show most of our results with α from 10 cycles to 20 cycles. The redundant encoding technique enables linear scaling with the increase of αs so that we can accommodate loose wearout bounds with a linear increase of the number of NEMS switches in the architecture. For the parameter β, we try various values from 4 to 16 according to the Weibull modeling of various kinds of devices in the literature. For example, as mentioned in Section 3.2.2, MEMS devices in [56] have βs of 12.94, 7.2, 8.58 with geometrical variations, material elasticity and resistance variations. According to [69], typical β values for MEMS reliability fall in 0.5 to 5. We try to push β values down (eg. 4) with redundant encoding to tolerate more process variations.

Then we study how many such devices the limited-use connection requires to meet the fast degradation requirement. The results are shown in Figure 3.4a without redundant encoding. With small αs, the NEMS switches have tight wearout bounds so that small parallel structures can meet the fast degradation requirement. With large αs, the number of NEMS switches increases exponentially to compensate with the loose wearout bounds (The y-axis in Figure 3.4a is in log scale). Similarly, with large βs, the NEMS switches are relatively consistent in the degradation so that small parallel structures are feasible. With small βs, the number of devices increases dramatically to control the variations.

Given the number of devices needed in the architecture, we estimate the area cost analytically assuming an H-tree layout of the NEMS switches and wires. The contact area of each NEMS switch is assumed to be $100nm^2$ and the distance between switches is assumed to be $1nm$ [33]. The area cost of representative engineering options are summarized in Table 3.1. Although loose wearout bounds and process variations can be compensated by investing more NEMS switches in the architecture to minimize the fabrication cost, the correspondent area cost also increases significantly. For example, when α is 18.69 and β is 10, the area cost is $0.52mm^2$, which could be hard to afford especially when we need to deploy the security hardware on mobile devices. In the next section, we demonstrate that we can reduce the area cost using redundant encoding techniques.

Table 3.1: Area cost of the limited-use connection

(α, β)	without encoding (mm^2)	with encoding $k = (10\% * n)$ (mm^2)
(10.51, 16)	1.27e-4	3.2e-5
(10.21, 10)	2.03e-3	1.3e-4
(19.68, 16)	2.03e-3	1.3e-4
(18.69, 10)	5.2e-1	1.3e-4

Redundant encoding

As discussed earlier, with redundant encoding, we enforce at least k working NEMS switches in each parallel structure in order to successfully decrypt the key that is required to validate the passcode. This encoding technique helps tighten the wearout bounds for each parallel structure even with NEMS switches that have relatively loose wearout bounds.

Figure 3.4b shows the effect of encoding the parallel structures with different levels of redundancy. The total number of NEMS switches needed decreases dramatically and scales linearly rather than exponentially with the increase of device wearout bounds. For example, without encoding, when α is 14 and β is 8, the number of NEMS switches needed in the architecture is about 4 billion. However, if with redundant encoding and $k = (10\% * n)$, the number of NEMS switches needed is only about 0.8 million with the same α and β. So, the redundant encoding helps reduce 4 orders of magnitude in the total number of NEMS switches. Moreover, it also improves the system tolerance to higher device variations with $\beta = 4$. With $k = (30\% * n)$, however, the decrease in the number of devices needed is negligible so that we can stop enforcing more requisite components in the encoding and decoding.

We assume the storage for component keys should be proportional to the size of the parallel structure and we accommodate that in the area cost evaluation in Table 3.1. We do not need extra logical circuits for the encoding/decoding because they can be done in CPU.

The switching energy is proportional to the size of each parallel structure. Assume the energy cost for each operation in NEMS switches is 10^{-20} Joule [33]. When α is 14, β is 8 and $k = (10\% * n)$, the total number of NEMS switches is 0.8 million and each parallel structure has 141 NEMS switches. Then the energy cost for each access is 1.41e-18 Joule. Since we use parallel structures, the switching time for each access equals individual NEMS switch's switching time, which is around 10 ns [33].

System-level access bounds

If small variations of the system-level access bounds are desired, we can tune the fast degradation criteria to achieve the variation. The fast degradation criteria in previous experiments are defined as follows: Each parallel structure has at least 99% probability for t accesses while at most 1% probability for $t+1$ accesses. The reliability of the lower bound can be extended to 99.99999% (for exponential decrease in the failure probability) with 3x linear increase in NEMS switches using redundant encoding, in which the minimum 91,250 accesses are guaranteed.

More interestingly, if the application has high tolerance to the upper bound of accesses, then, in each copy, the degradation criteria for $t+1$ accesses, p, can be relaxed from 1% to 10%, for instance. We analyze the number of devices needed and the empirical access bounds when using different degradation criteria, as shown in Figure 3.4c. When we increase p from 1% to 10%, the empirical access upper bound increases from 91,326 to 92,028 accesses, while the total number of NEMS switches needed is reduced by 40%.

Furthermore, if the passcode is sufficiently secure for many more attempts, we can extend the upper bound to the minimum guesses needed to crack the passcode. According to [57], only the most popular 1% passwords can be guessed with at least $100,000$ attempts. Similarly, 2% most popular passwords can be guessed with at least $200,000$ attempts. We need at least 675,250 NEMS switches to architect the limited-use connection with an upper bound of 91,326 accesses when $\beta = 8$ and $k = (10\% * n)$, according to Figure 3.4b. However, with upper bound targets of $100,000$ and $200,000$ accesses, we only need 38,325 and 29,200 NEMS switches, respectively, as shown in Figure 3.4d.

3.5 A limited-use targeting system

Similar to the security enhancement in smartphones, targeting systems can also benefit from the physically enforced limited-usage of targeting commands. Political alliances change over time and devices should be used only for the immediate mission.

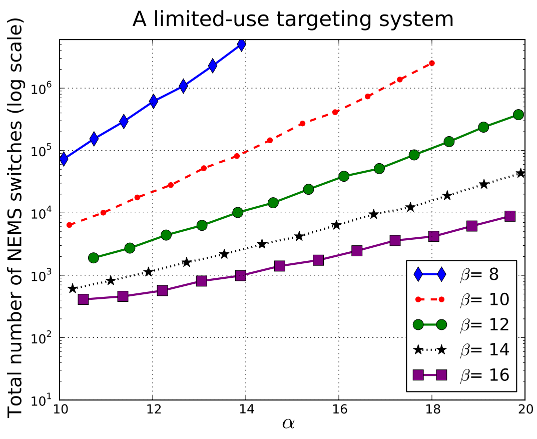
Targeting systems are usually composed of three functional subsystems: the command and control system, the communication network, and the launching station. The command

and control system makes targeting decisions according to the real-time data from radars and these orders will be encrypted and transmitted securely to the launching station through the encrypted communication link. Although targeting systems have been designed with a high priority for security, there are still many vulnerabilities. Cyber attacks are reported that the hacker can execute commands on a targeting system remotely by either gaining access to the control system or the real-time communication network [70].

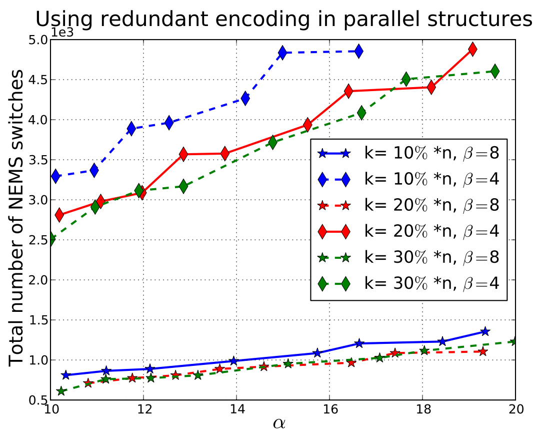
We propose to enhance the security of targeting systems with limited-use security architectures. Since the launching station is remotely operated through an encrypted communication link, we restrict the maximum attempts to decrypt the targeting commands at the launching station, which enforces an upper bound to the execution of the targeting commands. By enforcing the upper bound, we can enhance the security of the targeting system from two aspects: 1) it prevents excessive usage of the targeting system beyond the original task, 2) it prevents brute-force attacks to crack the encryption system.

As demonstrated in the limited-use connection use case, device wearout can help physically enforce the access bounds. Similarly, given an expected usage of the targeting commands in one task, such as 100 times, we can exploit the device wearout to build an architecture that automatically wears out after the 100th access to the command decryption key. And we build the architecture inside the launching system so that every access to the command decryption key needs to traverse through the architecture. Compared with the limited-use connection use case, the access bound here is relatively small that a small number of parallel structures are feasible. And the degradation criteria of the parallel structures should be strict because we do not want a single unintentional targeting command to be executed.

As a result, our design principles of using device wearout to build a limited-use targeting system are: 1) the targeting system should work reliably for the expected number of usage, 100 times, for instance. 2) the targeting system should not work for the 101st time. Since the design goals here are similar as in the last use case, we skip the detailed discussion of design



(a) Without redundant encoding



(b) With redundant encoding

Figure 3.5: The total number of NEMS switches needed with different engineering options for the limited-use targeting system.

options. Figure 3.5 shows the total number of NEMS switches needed with different engineering options. Since the targeted access bound is relatively small, the number of NEMS switches needed in the architecture is also reduced by several orders of magnitude compared with the limited-use connection use case. Without redundant encoding, the limited-use targeting system needs at least 8,855 NEMS switches with $\alpha = 20$ and $\beta = 16$. The worst case in Figure 3.5a is 842,941 NEMS switches with $\alpha = 14$ and $\beta = 8$. With redundant encoding, the total number of NEMS switches can be reduced to 810 when k=(10%*n), $\alpha = 10$ and $\beta = 8$, as shown in Figure 3.5b. The curves are less smooth because of the small usage target. Only 5 to 10 parallel structures are needed in total and small variations in device wearout bounds can change the total number of parallel structures.

3.6 Using device wearout to build one-time pads

One-time pads [71] are important cryptographic algorithms used in many important scenarios as they can provide perfect secrecy [72, 73]. However, to guarantee the perfect secrecy, important rules for one-time pads include that there must be only two copies of the keys (one for the sender and one for the receiver), one copy should be securely transmitted from the sender to the receiver before message transmission, and the sender and receiver must destroy each key immediately after each message encryption/decryption [74].

Traditionally, secret keys were written in real paper pads. Paper, however, severely limits the bandwidth of key distribution. One approach would be to use read destructive memories, which can share the keys and destroy them after use. They can not, however, resist adversarial cloning. Attackers may clone the one-time pads and make two copies, one copy to replace the receiver's original one and another copy for themselves to break the encryption in future message transmission between the sender and receiver (the archaically named "evil maid attack"). Another approach would be to use Physically-Unclonable Devices (PUFs) [75, 76] to fabricate an unclonable one-time pad. PUFs depend upon process variations in each chip, however, making it difficult to fabricate two identical chips so that a sender and receiver could share the pad. We need to both defend against stealthy replications by making it difficult for attackers to ever access the secret keys, yet offer reliable secret sharing between the sender and receiver.

We propose to use wearout devices to provide hardware enforced security for one-time pads. In Section 3.6.1, we use decision trees to distribute large random keys as a form of randomness amplification. In Section 3.6.2, we build hardware decision trees with NEMS switches to physically enforce the one-time usage of the keys. In Section 3.6.3, we exploit redundant encoding techniques to guarantee that the receiver can reliably retrieve the key but adversaries can not. The impacts of engineering options such as decision-tree heights, device wearout characteristics are discussed in Section 3.6.4.

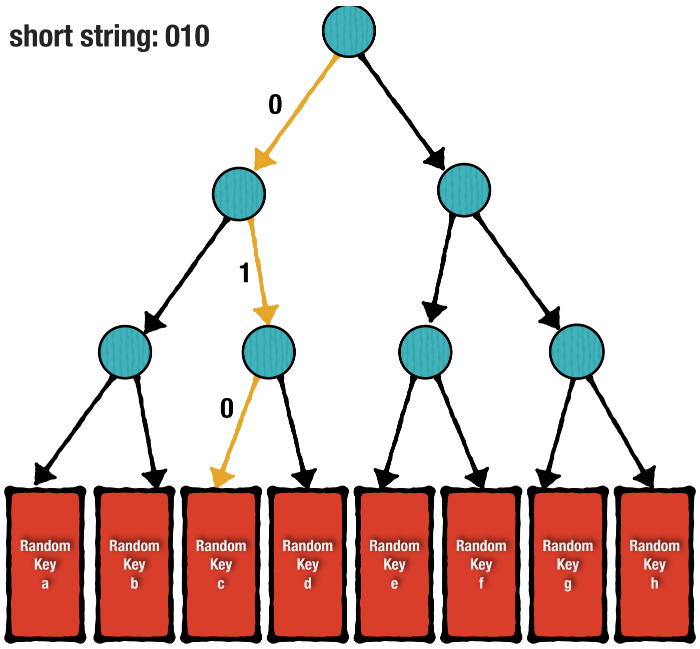


Figure 3.6: The decision-tree structure for randomness amplification. The receiver follows the short string to get the random key. At each branch, ‘0’ means left and ‘1’ means right. For instance, the short string “010” directs to random key c.

3.6.1 Secure transmission of large random keys

In one-time pads, each message employs a new key and the key must be at least as long as the message. As a result, large keys are required for long messages and these keys must be transmitted securely before any message transmission. To relax the requirement for secure transmission of the whole block of random keys, we design decision trees that store many potential keys in their leaves and each key is indexed by a short string about the path information, as illustrated in Figure 3.6. We assume only the sender and receiver share the right path so that adversaries can only do random path trials to obtain the secret keys.

As a result, the secure transmission of large random keys is divided into the secure transmission of two parts: a short path string and a decision tree that contains many potential random keys. On one hand, the secure transmission of the short path string is less expensive compared with the whole block of random keys. There are many choices for the transmission media and people can even memorize it. If using a temporary channel for the short string, there is less opportunity for adversaries to break it since the transmission time could be very short. On the other hand, decision trees are implemented in wearout devices on a chip, and the secure transmission of them is guaranteed by our design with NEMS devices, which will be explained in Section 3.6.2.

The chip that contains many decision trees (many random keys) is our new set of "one-time pads" that should be delivered to the receiver beforehand for many instances of potential message transmission. Even if the chip is obtained by adversaries, the adversaries still have little chance obtaining the right random keys without the right path information. Moreover, the decision tree will be destroyed very quickly after one trial because of the underlying wearout devices. We will explain the hardware implementation of decision trees in detail in Section 3.6.2 and Section 3.6.3.

3.6.2 Hardware design of one-time decision trees with NEMS switches

Design principles

To guarantee the security of one-time pads in decision trees, we need to follow the rules that random strings must be transmitted securely (without stealthy cloning) and destroyed immediately after use. In general, the hardware design of decision trees using NEMS switches is guided by the following principles:

- At least one path should work so that the receiver is able to use the key at least once.
- Not many paths should work, which will prevent adversaries from getting the key within limited trials.
- Each tree should be area efficient so that we can maximize the density of one-time pads on a fixed-size chip.

We build the decision-tree circuit with NEMS switches as intermediate nodes of the tree. On

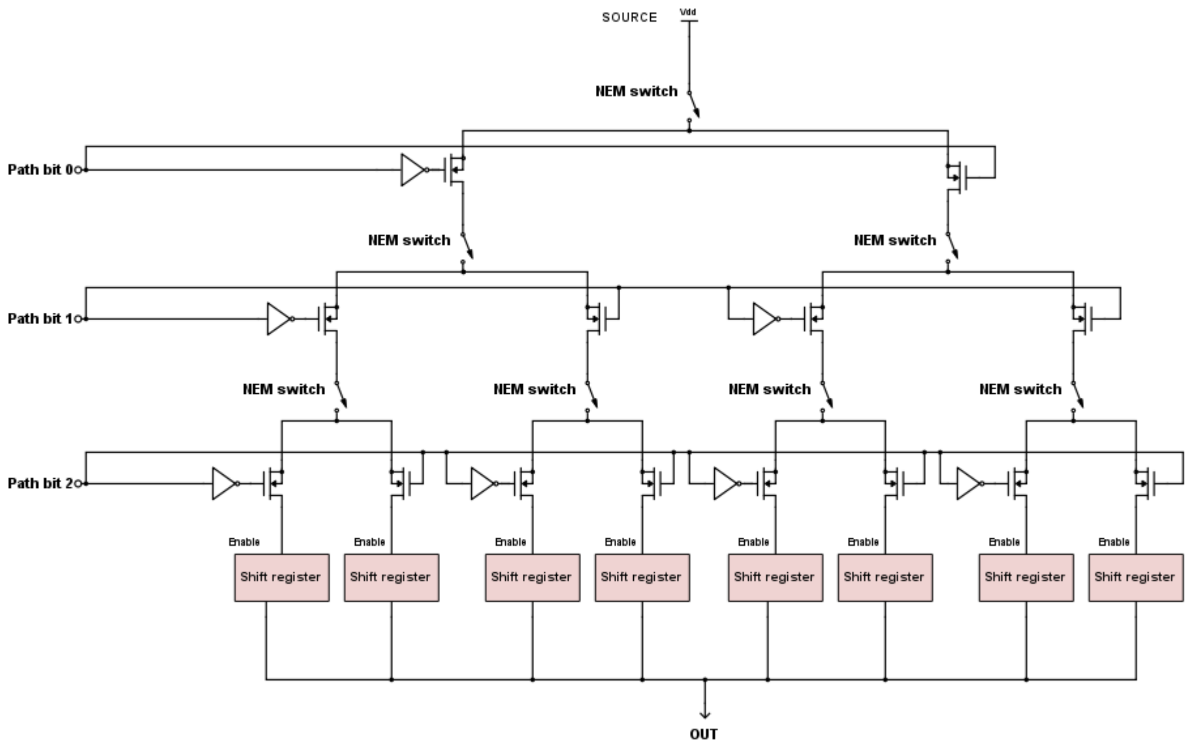


Figure 3.7: Schematic graph of the implementation of a 3-layer decision tree using NEMS switches

one hand, to allow the receiver to use the tree at least once, the path to the right random string should be successful at least for the first time, which imposes a lower bound on the path reliability. On the other hand, to effectively withstand adversaries' attacks, the paths should fail as quickly as possible so that not many of the paths work successfully, which imposes an upper bound on the path reliability. Similar to the first two use cases, we can physically enforce the access bounds by carefully engineering the decision-tree structures with NEMS switches.

Hardware design of decision trees

Figure 3.7 shows the schematic of the decision-tree circuit. The decision-tree circuit uses the short path string as control bits to open up the path to the right random string and sends out the right random string serially. The intermediate nodes in the decision tree are implemented with NEMS switches that wear out very quickly, while the random keys in the leaves are implemented in read destructive shift registers. Simply relying on an array of read-destructive or one-time programmable devices (eg. anti-fuse technologies), however, would be vulnerable to "evil maid attacks". The read-destruction could be compromised if reading with a lower voltage. Attackers could also easily clone the devices and bypass the destruction. Our architecture with NEMS switches on the paths will resist cloning by making it difficult for attackers to get access to the memory. And we distribute the random keys into many small memory devices so that it will be hard for attackers to inspect all of them.

As shown in Figure 3.7, one NEMS switch is used at each branch of the decision tree. Only if all the intermediate nodes along the right path survive after the first access can the receiver successfully obtain the target random string. If each NEMS switch could only be accessed once ideally, then accessing each intermediate node should allow you to choose one path but destroy the other one at the same time since the intermediate switch would fail next time. As a result, only one access could be made to the ideal decision tree and this decision tree meets all of our design goals. However, practical NEMS switches are hard to provide deterministic wearout bounds due to fabrication and process variations. As a result, we exploit the probabilistic modeling of NEMS switches, discussed in Section 3.2, to reason about the security of hardware decision trees.

Probabilistic reasoning

According to the reliability model in Equation (3.3), the probability of each NEMS switch surviving the first access is $R(1) = e^{-\left(\frac{1}{\alpha}\right)^{\beta}}$. Assume the height of the decision tree is H. Then the probability of successfully getting through the right path is $(R(1))^H = e^{-\left(\frac{1}{\alpha}\right)^{\beta} H}$. If any of the NEMS switches on the right path failed for the first access, then no one would ever get to the right random key.

A successful decision-tree design should enable a successful first access but prevent any subsequent accesses. The probability for a successful second trial can be throttled by designing a high tree or using NEMS switches with tight wearout bounds so as to guarantee the security and one-time usage of the decision tree. However, under this condition, the probability of a successful first access is also restricted. We should guarantee that the receiver can succeed at least once without sacrificing the security at the same time.

One solution to improve the one-time pads' reliability is to provide multiple copies of the same decision tree for each transmission. The receiver can get the key as far as the right path in one copy is successful. However, the challenge of this solution is to avoid leaking information to adversaries with multiple copies of the same random keys. We solve this problem by exploiting Shamir's secret sharing mechanism and redundant encoding, as discussed in Section 3.6.3.

3.6.3 Redundant encoding for reliable and secure key transmission

To prevent information leakage to adversaries with redundant copies, we encode the random strings with error-correction codes and spread them into different copies. Some copies may be erased because of the device failures. The receiver can recover the right random string with a small number of failed copies. The desired feature of the error correction codes, however, is fast degradation once we get more failures beyond our error tolerance target to withstand adversaries' attacks.

Based on Shamir's secret-sharing scheme, as described in Section 3.4.1, each random string is encoded into n component keys stored at the same position of n copies of the decision tree: S_1, S_2, ..., S_n. Each random string S can be computed with k or more S_i components. As a result, to guarantee both the reliability and security of random keys, we need to guarantee that the receiver has close to one probability to get through k or more paths successfully, while adversaries have close to zero probability to do the same. The main difference between the receiver and adversaries is that we assume adversaries have no knowledge about the right path information.

Probabilistic modeling of successful one-time pads

Given the redundant encoding technique, we can calculate the receiver's success probability and adversaries' success probability analytically. The receiver's success probability on one copy is:

$$S_{recv}^{(1)} = e^{-\left(\frac{1}{\alpha}\right)^{\beta} H} \tag{3.9}$$

The probability of getting at least k out of n copies successfully is:

$$S_{recv}^{(k+)} = \sum_{i=k}^{i=n} \binom{n}{i} \left(S_{recv}^{(1)}\right)^{i} \left(1 - S_{recv}^{(1)}\right)^{(n-i)} \tag{3.10}$$

For adversaries, we first consider that they can get through x paths successfully out of n copies. Then, we need to calculate the success probability of k or more out of x being the correct path, with the probability of each successful path being the correct path as follows (since there are $2^{(H-1)}$ paths in total):

$$P = \frac{1}{2^{(H-1)}} \tag{3.11}$$

The adversaries' success probability for getting through one path in one copy is:

$$S_{adv}^{(1)} = e^{-\left(\frac{1}{\alpha}\right)^{\beta} H} \tag{3.12}$$

The probability of getting through x paths in n copies is:

$$Prob(x) = \binom{n}{x} \left(S_{adv}^{(1)}\right)^{x} \left(1 - S_{adv}^{(1)}\right)^{(n-x)} \tag{3.13}$$

The probability that k or more out of x successful paths are the right paths is:

$$Prob_x(k+) = \sum_{i=k}^{i=x} \binom{x}{i} P^{i} (1-P)^{(x-i)} \tag{3.14}$$

The probability of getting at least k out of n copies successfully for adversaries is:

$$S_{adv}^{(k+)} = \sum_{x=k}^{x=n} \left(Prob(x) Prob_x(k+)\right) \tag{3.15}$$

, in which $k \leq x \leq n$.

The engineering goal for one-time pads in decision trees is to make sure that $S_{recv}^{(k+)}$ is close to one and $S_{adv}^{(k+)}$ is close to zero. Parameters in the above equations such as H, α, β, n, k will be discussed in Section 3.6.4 for trade-offs among fabrication cost, area cost, encoding complexity, etc.

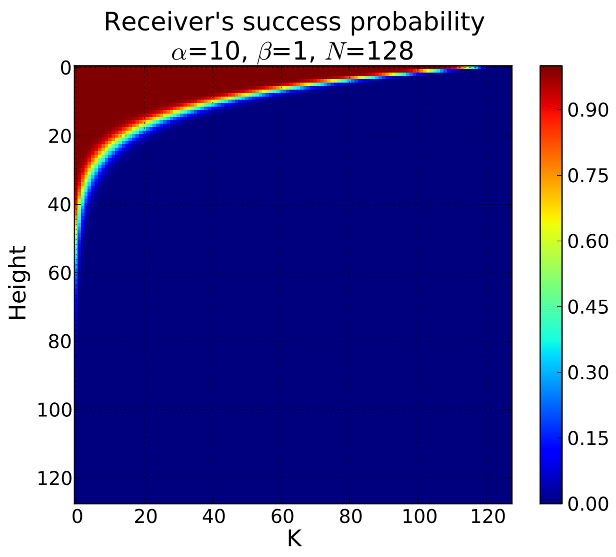
3.6.4 Engineering space exploration

In this section, we explore the engineering space that can 1) guarantee the success of one-time pads (in both security and reliability) and 2) reduce the fabrication cost and area cost.

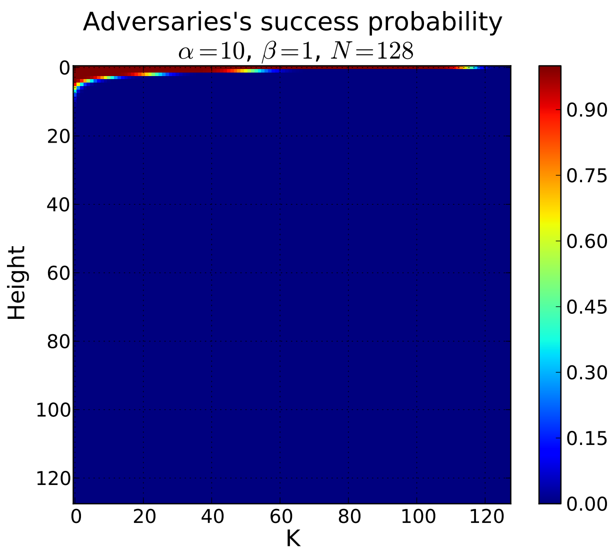
We first use a specific type of NEMS switch with an expected lifetime of 10 cycles: $\alpha = 10$ and $\beta = 1$. We can deal with high process variations (small βs) because only the reliability of the first access can affect receiver's and adversaries' success probability. For each key transmission, we use 128 copies of the same decision tree ($n = 128$).

Redundancy levels

Redundancy levels and encoding complexity will be reflected in the values of k. The receiver's and adversaries' success probability with different ks and tree heights Hs is presented in Figure 3.8. The intersection of the red area in Figure 3.8a and the blue area in Figure 3.8b is the success space for one-time pads. With high redundancy, the secret is easy to recover. In contrast, with low redundancy, the access to the secret becomes more difficult since only receiving enough components can help recover the secret. As a result, low redundancy leads to high security. As shown in Figure 3.8a and Figure 3.8b, both receiver's and adversaries' success space shrink quickly with the increase of k (decrease of redundancy) but adversaries fail faster. Moreover, when both n and k increase, the complexity of encoding and decoding random strings increases because the latency for constructing and solving the polynomial systems increases.

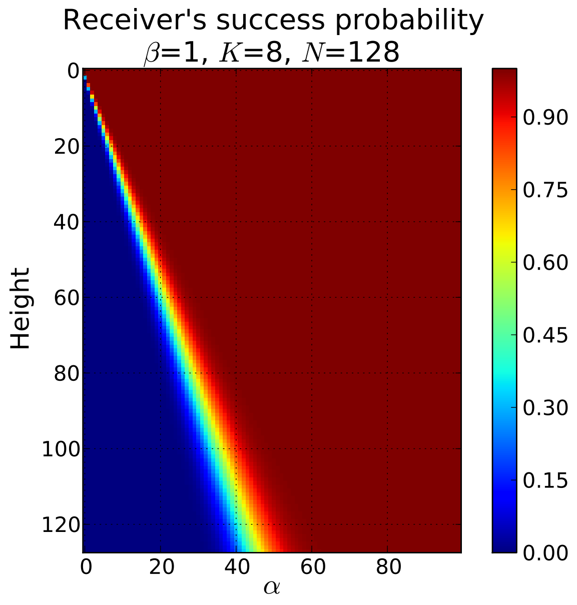


(a) receiver

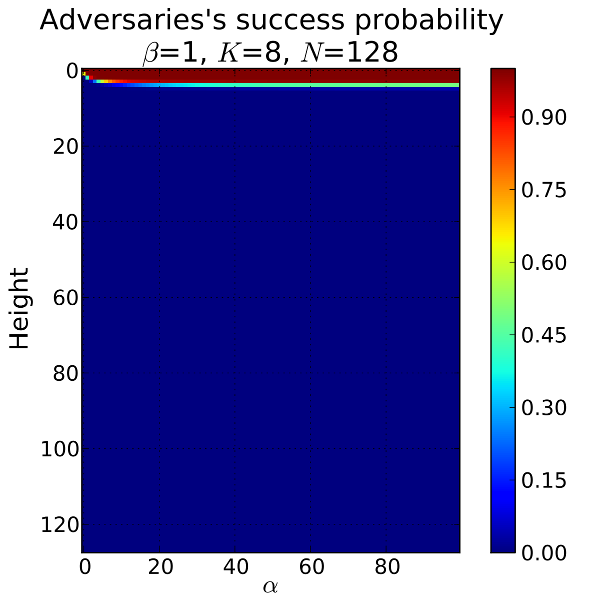


(b) adversaries

Figure 3.8: Success probability with different tree heights and ks (redundancy levels). The intersection of the red area in the left figure and the blue area in the right figure is one-time pads' success space for both reliability and security.



(a) receiver



(b) adversaries

Figure 3.9: Success probability with different wearout bounds (MTTFs) and tree heights.

Tree heights

Higher trees can also enhance the security of one-time pads because 1) the path to the random keys gets longer so that the probability of getting through all the nodes along the path for both the receiver and adversaries becomes smaller, and 2) the number of paths increases exponentially so that it is even harder for adversaries to get on the right path. In Figure 3.8a, the receiver's success area shrinks when the tree height increases. However, in Figure 3.8b, the tree height can effectively block adversaries. When the tree height is 8 or more, the adversaries' success probability reduces to zero even if the redundancy level is very high (k is close to 0).

In summary, redundancy provides reliability for the receiver and the tuning of redundancy levels can trade encoding/decoding performance for security. And higher trees can be exploited to further improve the security of one-time pads, while with higher area and performance cost.

In Figure 3.8, we used a specific type of NEMS switch in which the mean time to failure is about 10 cycles. This can achieve a large success space for one-time pads. However, such NEMS switches may be expensive to fabricate to enforce the wearout fast and under control. Figure 3.9 shows the impacts of different device wearout bounds defined by the mean time to failure or α values. With higher αs, both the receiver and adversaries have a higher probability of getting the right key. To ensure security, we need higher trees or less redundancy to compensate for the loose wearout bounds of devices. We can see the trade-off between tree heights and wearout bounds when $H \leq 7$: higher trees compensate for looser wearout bounds. When $H \geq 8$, the tree height can effectively withstand any adversaries' attacks. Lower device variations (larger βs) lead to smaller wearout windows. When the target access bound is only one cycle, larger βs postpone the wearout so they do not help ensure security, which also indicates high tolerance to process variations while offering high reliability for the sender and receiver.

3.6.5 Evaluation

Given the design of hardware one-time pads in decision trees, we want to evaluate how many times a single $1mm^2$ chip can be used for message transmissions between the sender and receiver. We also want to evaluate the latency and energy cost for retrieving a key each time.

Density estimate

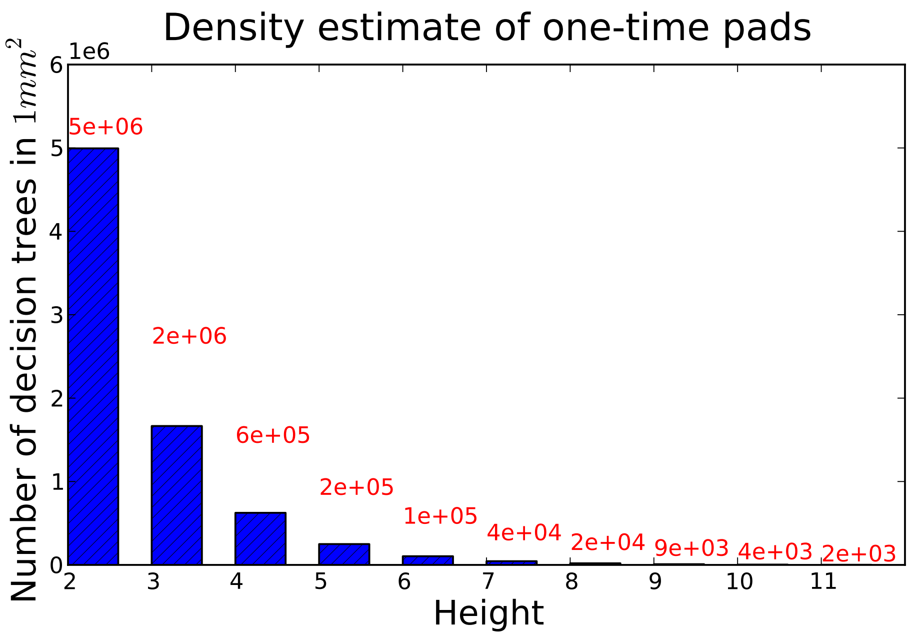


Figure 3.10: Density estimate

We assume an H-tree layout for the decision tree circuit. The area cost of a complete binary tree in H-layout is at the order of the number of leaves in the tree if nodes are separated with unit distance [77]. Given that the height of each decision tree is H, there are $2^{(H-1)}$ leaves. We assume $100nm^2$ for the dimension of each NEMS switch [51, 78, 33]. Then the area cost for the decision tree is about $100 * 2^{(H-1)}nm^2$. The area cost for shift registers is linear to the size of random strings. The size of each random string is assumed to be proportional to the tree height, that is, around $1000H$ bits. Then the area cost for all the shift registers is $2^{(H-1)} * 1000H * 50nm^2$, assuming a $50nm^2$ cell in the registers.

The number of decision trees we can accommodate in the chip is calculated, as shown in Figure 3.10. If $H = 4$ and $N = 128$, then we can transmit around $4,687$ one-time pads in this chip.

Latency and energy cost

The latency to retrieve a random key is proportional to the path length and number of decision-tree copies: $lat = \alpha HN$, with α being the delay of a single NEMS switch, at the order of 10 ns [33]. If $N = 128$ and $H = 4$ as in our previous example, the latency to get to the random strings is around 0.00512ms in the worst case. The latency for reading the random string out from the shift register is proportional to the length of the random string. We assume that the propagation delay per bit is around 20ns, such as in MM74HC165 parallel-in/serial-out shift registers. Then the delay of reading the whole random string out is about 0.08ms ($20ns * 1000H$). So the total delay for each random key retrieval is around 0.08512ms.

Similarly, the energy cost on the path to the random string is 5.12e-18 ($NH * 10^{-20}$) Joule in the worst case. The energy cost in reading the random strings out is negligible since only one read from a shift register is needed after successfully getting through a path.

3.7 Limitations

This is an initial work to exploit a contrarian view of wearout to build limited-use security architectures. However, more experimentation and discussion on NEMS failure models and security mechanisms are needed in the future.

The primary limitation of the current work is that, even with techniques to decrease system-level sensitivity to device variation, device parameters must still fall within a specific range to make system use targets practical. Note also that we reduce sensitivity to the scale parameter (that represents devices' mean time to failure) but not the shape parameter (which represents devices' variations in lifetime degradation). Furthermore, although the Weibull model is highly parameterized, we need experimental data to validate the range of parameters that are realistic of this or other alternative models. Finally, we might compromise the device's availability for legitimate users in order to guarantee strong confidentiality and integrity. An attacker could purposely degrade the NEMS network through many password guesses to consume the legitimate usage bound, which will hurt the availability for legitimate users. The key issue, however, is to guarantee confidentiality and integrity of the data after the attacker got access to the device. In our design, intentional consumption of the legitimate usage bound could only degrade the device faster, but not leak any information of the confidential data.

3.8 Related work

Hardware security has attracted a lot of interests recently. As mobile and embedded devices become ubiquitous, attackers could easily get access to the physical devices. In order to achieve confidential information in the devices, they could exploit any hardware measures to crack the devices, such as brute-force, reverse engineering, side channels, etc [79]. Researchers have proposed a variety of countermeasures to defend against such kinds of physical attacks.These countermeasures can be generally classified into two taxonomies: software assisted approaches and pure hardware approaches:

Software assisted approaches Many existing hardware security solutions involve the cooperation of software and hardware [80, 58]. For example, modern smartphones have tamper-resistant hardware modules (eg. SoC's secure enclave) to protect the processing and the storage of the device's encryption key and user's confidential information, yet some important security policies that restrict the usage of the keys are implemented in software. The software interface, however, could expose security vulnerabilities and compromise the hardware security. Software may have bugs that lead to violations of the policy. The device could be reprogrammed, either by adversaries or under duress, through these software APIs to implement a different policy. Hence, pure hardware solutions are desired to physically secure the confidential data.

Hardware approaches Hardware approaches exploit physical device characteristics to protect the data. The specific hardware measures have been ad-hoc and we summarize them into the following three taxonomies:

1. Physical disorder based security: Physically unclonable functions (PUFs) and random number generators (RNGs) are most popular hardware security primitives in this category [81]. They are based on the inherent randomness in each device due to process variations to generate a unique key for each device for device authentication, IP protection, random number generation, etc. A variety of hardware technologies have been explored to provide the physical disorder, such as SRAM [82], DRAM [83], memristors [84, 85], nanotechnologies [86], etc.

2. Physical degradation based security: This taxonomy refers to hardware measures that enforce physical usage bounds through purposely degradation of the hardware. [87] created the first self-enforceable hardware for software and content usage metering. They employed the aging effects in transistors due to negative bias temperature instability to measure the time a particular licensed software is used. Similarly, [88] used the SRAM decay phenomenon to measure time for batteryless embedded devices in order to throttle response rates to adversarial accesses. Recently, [89] proposed a memristor-based neuromorphic computing system that can resist adversarial learning of the model and data by degrading the learning accuracy nonlinearly after more inputs are applied to the learning algorithm. Our proposal also stems from this taxonomy but we tailor the devices' degradation characteristics to meet the system level usage requirements through a series of techniques. The methodology can be generally applied to many security architectures where a physically enforced usage window is desired.

3. Physical destruction based security: Some other hardware approaches can restrict data accesses through self-destructing circuits. However, most current self-destructing devices can

only destruct the data, but not the devices [90, 91]. By reprogramming or cloning the devices, all the internal data could be cloned into multiple copies so that the data cannot be secured. Other self-destructing devices need external operations to trigger the destruction [92, 93, 94]. For example, DARPA displayed a new chip built on strained glass substrates that can shatter within 10 seconds when remotely triggered [94]. Nevertheless, our system wears out automatically without a need for remote control.

Meanwhile, more research efforts are needed in the future to study the general design principles, the evaluation and verification methodologies of security architectures that can offer hardware-enforced security.

3.9 Chapter Summary

In this work, we propose methodologies of using wearout devices to build security architectures. We explored a probabilistic wearout model with Weibull distribution to characterize the behaviors of NEMS wearout. Based on these characteristics, we design architectures that can physically limit attacks while accommodating legitimate usage. Three use cases are examined: a limited-use connection, a limited-use targeting system and one-time pads. We first present a family of architectural techniques to meet minimum and maximum system-level usage bounds and characterize the design space in terms of device variability (which affects fabrication cost) and device count (which affects area and power). Then we use redundant encoding techniques to improve the security architectures from exponential scaling to linear scaling with the increase of device wearout bounds in the limited-use connection and limited-use targeting system use cases. In the use case of one-time pads, the redundant encoding (Shamir's secret sharing scheme) can effectively throttle the possibility of leaking secret information to adversaries and thus guarantee both reliability and security of one-time pads.

Overall, we envision new opportunities for physically limiting vulnerability to attacks through careful engineering of intentional device wearout.

Chapter 4

Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Tradeoffs in NVMs

4.1 Introduction

The traditional memory systems based on DRAM technologies have been facing increasing challenges due to DRAM scaling issues. Non-volatile memories (NVM), both commercialized (e.g., NAND Flash [95]) and emerging ones (e.g., PCM [96, 97], ReRAM [98] and NEMS [33, 99]), are considered as promising replacements for DRAM. Compared with DRAM technologies, these NVM technologies offer persistence, much higher scalability and lower stand-by power and non-volatility. However, there are also disadvantages of these NVM technologies. Here are two of the most common ones:

- **Performance.** The write/read access latencies of these technologies are considerably longer than DRAM's. As a result, NVM has lower performance than DRAM.

- **Lifetime.** These NVM technologies usually have limited write endurance [100][32][30][101]. If without special treatment, NVM will have short lifetime because (some of) the memory cells will be worn out soon. As a result, special mechanisms (eg., wear leveling [30] and/or wear limiting [97]) must be used to guarantee the lifetime of NVM memories.

Various techniques have been proposed to combat the performance and lifetime issues of NVMs [102, 32]. Unfortunately, the goals of boosting performance and prolonging lifetime are often in opposition. For example, write cancellation [1][103], which is an effective technique to improve the NVM performance by prioritizing the reads over writes, results in extra writes to the NVM, thus shortening the lifetime. On the contrary, using slower and less destructive writes [32] can improve the NVM lifetime, but at the expense of lower memory system performance.

Intuitively, we can use a combination of techniques (some for lifetime improvement, some for performance improvement) to achieve a sweetspot between performance and lifetime. However, there are several practical issues to achieve this goal:

- **Huge configuration space.** The whole configuration space is huge not only because it may contain multiple techniques, but also because each individual technique itself contains multiple configurable parameters to control its aggressiveness. For example, in our experiment, the total configuration space includes 3,164 configurations, which is magnitudes larger than the configuration space of prior arts [104, 105, 106].

- **High sensitivity to applications.** The impacts of some techniques are very sensitive to different applications. As a result, the ideal combination of techniques for different applications are dramatically different, as will be shown in Section 4.3.3. They differ not just in the choices of techniques, but also in the aggressiveness of each chosen technique.

- **User-defined objectives.** Different from several prior proposals that aim at improving a single well-defined goal (e.g., IPC) [104, 106], the desired tradeoff for the NVM changes in different situations. This is especially true when it comes to the lifetime of NVM memory: some users want the memory to last longer (eg. 8 years) and they are willing to pay some performance penalty for it; and some other users want the computing system to run at faster speed even if its NVM system will break down after just few years (eg. 4 years). For these two optimization goals here, the ideal configurations are usually dramatically different.

To adapt the architectural techniques in NVMs to different scenarios, we propose a general, learning-based framework, *Memory Cocktail Therapy (MCT*), which tailors a specific combination of techniques for each application and user-defined objective function. Specifically, MCT formulates all different combinations of techniques into a high-dimensional configuration space and then employs machine learning techniques to model the behaviors of different configurations based on a small set of samples. Given the estimation of all different configurations, MCT selects the optimal configuration that satisfies the requirements specified by the user.

The key challenge in implementing the framework, however, is to select a near-optimal configuration with negligible overhead at runtime, and ideally with little modification to the hardware. We implement MCT so that it automatically reduces the dimensionality of the configuration space by lasso regularization and uses the selected features to guide runtime sampling, leading to more informative samples and higher prediction accuracies. Also, we choose lightweight, yet accurate online predictors such as quadratic regression and gradient boosting, to predict the behaviors of other configurations based on the samples. Furthermore, we implement MCT with a lightweight phase detector and fined-grained sampling technique to accommodate both coarse-grained and fine-grained phases in memory behaviors, which do not rely on prior knowledge about the phases or significant modifications in hardware (except performance counters).

In summary, our work has the following contributions:

- We formulate NVM system design problems with various tradeoffs as constrained optimization problems (e.g. maximizing performance under a lifetime constraint). We motivate the need for machine learning techniques because of the high dimensionality of the configuration space, the high correlation and nonlinear impacts of configurations and the heterogeneity among applications. To our knowledge, our work is the first to use machine learning techniques to solve such problems.
- Rather than applying machine learning as a black box, we first compare various machine learning models based on their prediction accuracy, computation overhead, convergence rate, etc., and choose the optimal ones. Then, we improve the performance of machine learning models by data normalization, regularization, feature selection, and training with informative sample configurations.
- In addition to machine learning techniques, we use architectural insight to improve our scheme's robustness. For example, we observe that the impact of *Wear Quota* [32] is difficult to predict. Thus, we exclude it from the learning procedures to achieve better accuracy and then use it as the last resort to ensure lifetime goals are met despite inaccurate predictions.
- The framework can also adapt to dramatic phase changes in memory behaviors by guiding the learning procedures with a lightweight phase detector.
- Finally, MCT manages to dynamically choose the near-optimal NVM configuration with no hardware modification and minimal runtime overhead. Compared to a NVM-based system with ideal configurations for different applications, MCT using gradient boosting achieves 94.49% of the maximum performance and consumes only 5.3% more energy.

Table 4.1: Trade-offs of NVM and Their Impacts on Performance and NVM Lifetime

Trade-offs	Impact on Performance	Impact on Memory Lifetime	Related Proposals
With or without Write Cancellation.	Using write cancellation usually improves performance.	Using write cancellation shorterns NVM lifetime.	[1][103][32][107][31], etc.
With or without Eager/Early Writeback.	Using eager/early writeback usually improves performance.	Using eager/early writeback shorterns NVM lifetime.	[108][109][32],etc.
Write Latency VS. Endurance.	Using long-latency-high-endurance writes degrades performance.	Using long-latency-high-endurance writes prolongs NVM lifetime.	[32][110], etc.
Write Latency VS. Retention.	Using short-latency-short-retention writes usually improves performance.	Using short-latency-short-retention writes shorterns NVM lifetime.	[111][112][113], etc.
Read Latency VS. Read Disturbance.	Using short-latency-high-disturbance reads usually improves performance.	Using short-latency-high-disturbance reads shortens NVM lifetime.	[114][115], etc.

The rest of this chapter is organized as follows. Section 4.2 introduces NVM architectural techniques and their impacts on performance and lifetime. In Section 4.3, we provide a case study of the optimization problem in mellow writes. Then we introduce our framework of *Memory Cocktail Therapy* in detail in Section 4.4 and its implementation in Section 4.5. The experimental methodology and results are presented in Section 4.6. Section 4.7 presents related work, and Section 4.8 summarizes the chapter.

4.2 Background

Various trade-offs exist in non-volatile memories which can be utilized to improve their performance or memory lifetime. However, in many cases, the trade-offs used to improve the NVM performance considerably degrade the NVM lifetime, and vice versa.

In this section, we introduce in detail several representative trade-offs for non-volatile memories and their impact on performance and NVM lifetime, as listed in Table 4.1.These trade-offs include:

- **With or without Write Cancellation.** Write cancellation [1][103] usually improves NVM performance because it lets read request be served sooner. However, it also degrades memory lifetime since it performs more writes to the NVM memory.

- **With or without Eager/Early Writeback.** Eager writeback [108] utilizes idle memory intervals to eagerly perform write operations of data in the last level cache (LLC) before their eviction, so there is less possibility that the write queue is blocked. As a result, it usually improves performance, However, it also degrades the NVM lifetime since some eagerly written back data need to be rewritten before their eviction.

- **Write Latency VS. Endurance.** The endurance of NVM [32] can be considerably improved if the write operations are performed with lower power and longer latency, thus the NVM lifetime will be significantly prolonged. However, it also significantly degrades the performance of NVM due to longer write latency.

- **Write Latency VS. Retention.** In Multi-Level Cell (MLC) NVM, a write operation usually consists of one RESET and multiple SETs. Shorter write latency can be achieved by reducing the number of SETs in a write operation, at the expense of shorter retention time which requries more frequent refresh writes/scrubs and thus degrades the NVM lifetime [111][112]. Some proposals also claim that there is a similar trade-off in Single-Level Cell (SLC) NVM [113].

- **Read Latency VS. Read Disturbance.** It is also possible to improve the NVM read performance by using short latency but high disturbance reads [114][115]. However, this also comes with NVM lifetime overhead, since such fast reads require frequent refresh/scrub the read NVM cells and thus degrade the NVM lifetime.

Based on these trade-offs, researchers proposed various techniques to improve the performance or lifetime of NVM memory. For example, Eager Writeback [108], Preset [109] and Eager Mellow Writes [32] techniques all utilize the trade-off of Eager/Early Writeback; also, a large amount of proposals (e.g., [107][31][103]) utilize the trade-off of write cancellation. Furthermore, to achieve a performance-lifetime sweet spot of the utilized trade-off, nearly all the

Table 4.2: Techniques of the evaluated combined technique.

techniques	**value**	**discrete parameters**	**continuous parameters**
Default	N/A	*fast_cancellation*	*fast_latency*
Bank-Aware Mellow Writes (*bank_aware*)	true /false	*slow_cancellation*	*slow_latency* *bank_aware_threshold*
Eager Mellow Writes (*eager_writebacks*)	true /false	*slow_cancellation*	*slow_latency* *eager_threshold*
Wear Quota (*wear_quota*)	true /false		*wear_quota_target*

Table 4.3: Parameters of the evaluated combined technique.

parameters	**value**
fast_cancellation	true/false
slow_cancellation	true/false (true if *fast_cancellation* is true)
fast_latency	[1, 4]
slow_latency	[1, 4] (greater than *fast_latency*)
bank_aware_threshold	[1, 4] (in entries per bank)
eager_threshold	[4, 32]
wear_quota_target	[4, 10] (in years)

Table 4.4: *Ideal* Configurations for different minimal lifetime constraint of application `leslie3d`

	bank _aware	*bank _aware _threshold*	*eager _writebacks*	*eager _threshold*	*wear _quota*	*wear _quota _target*	*fast _latency*	*slow _latency*	*fast _cancel- lation*	*slow _cancel- lation*
4.0 years	True	1	True	4	False	N/A	1.0	2.0	False	True
6.0 years	False	N/A	True	4	False	N/A	1.5	2.5	False	True
8.0 years	True	1	True	4	False	N/A	1.5	3.0	False	True
10.0 years	True	4	True	4	False	N/A	1.5	3.5	False	True

proposed techniques (e.g., [1][109][32]) are involved with some configuration mechanism to control its aggressiveness. For each individual application, the ideal configuration of a proposed technique is usually different.

Ideally, if we combine multiple techniques together, the combined technique might achieve a better performance-lifetime balance than all the individual techniques. However, as will be shown in Section 4.3.1, the configuration space of the combined techniques will be magnitudes larger than the configuration space of each individual technique. As a result, it is challenging to find the ideal configuration among thousands of candidates.

Table 4.5: *Ideal* Configurations for different applications

	bank _aware	*bank _aware _threshold*	*eager _writebacks*	*eager _threshold*	*wear _quota*	*wear _quota _target*	*fast _latency*	*slow _latency*	*fast _cancel-lation*	*slow _cancel-lation*
default	False	N/A	False	N/A	False	N/A	1.0	N/A	False	N/A
baseline	True	1	True	32	True	8.0	1.0	3.0	False	True
lbm_ideal	True	4	True	16	True	8.0	1.5	3.0	False	False
zeusmp_ideal	False	N/A	True	8	False	N/A	1.0	1.0	True	False
bwaves_ideal	True	3	True	32	False	N/A	1.0	1.5	False	True
stream_ideal	False	N/A	True	4	True	8.0	1.5	1.5	False	False

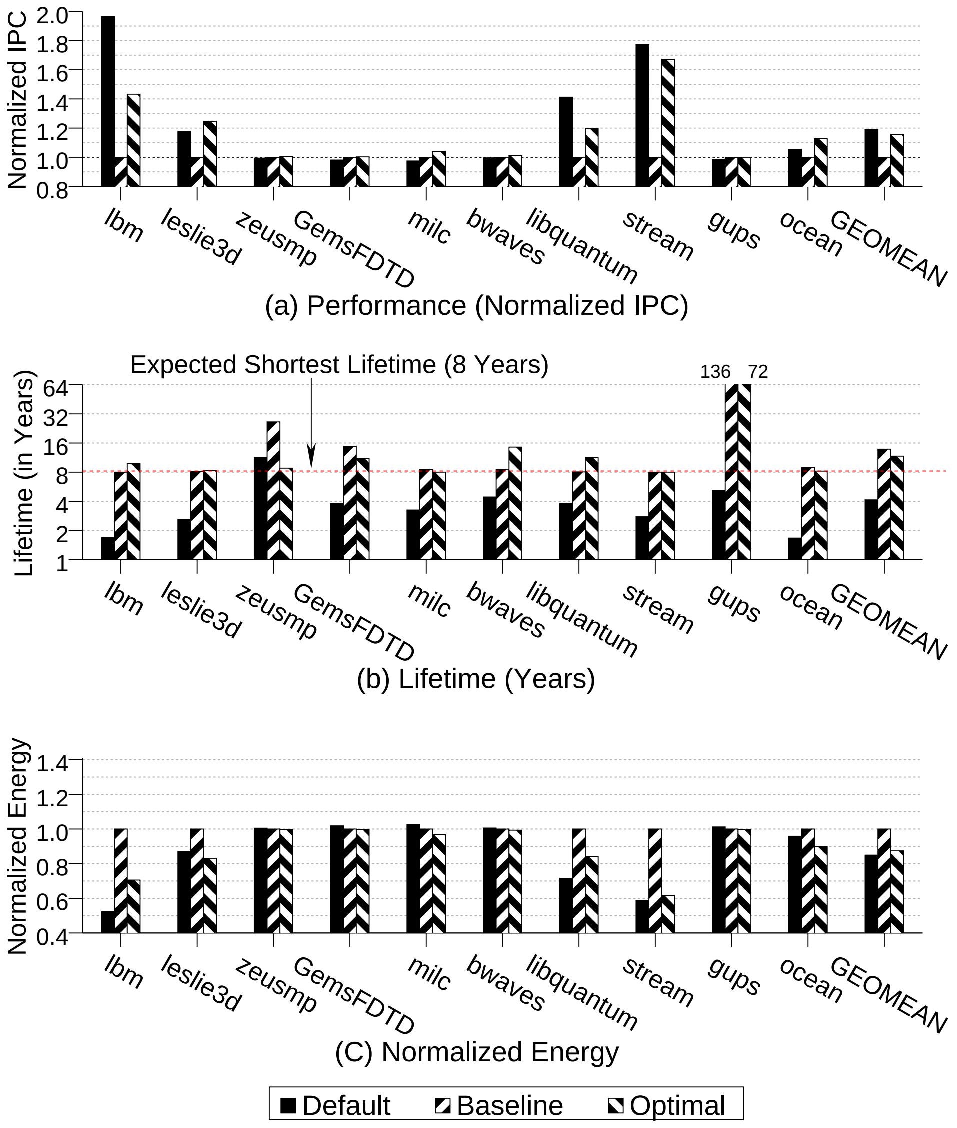


Figure 4.1: IPC, lifetime and energy comparison of *default*, *baseline* and *ideal* configurations.

4.3 Case study

4.3.1 Mellow writes configurations

As a case study, we focus on solving the optimization of *Mellow Writes* [32], which covers a series of techniques that utilize multiple tradeoffs in NVMs (eg. *write latency VS. endurance*, *write cancellation* and *eager/early writeback*) . In each technique, there are several configurable parameters that controls the usage of this technique and its aggressiveness. In general, *mellow writes* implements different write latencies and balances between performance and lifetime by carefully scheduling fast writes and slow writes based on the temporal and spatial patterns in the memory system.

- **Default.** *Normal* is the default technique which uses fast (normal) writes. There are two configurable parameters with the default technique: *fast_cancellation* and *fast_latency*. The former one indicates whether to use write cancellation for fast writes, and the latter one is the normalized speed used for fast writes.

- **Bank-Aware Mellow Writes (*bank_aware*).** This technique issues slow writes when the current memory bank is not *busy*. There are three parameters with this option: *slow_cancellation*, *slow_latency* and *bank_aware_threshold*. The first parameter indicates whether to use write cancellation for slow writes. The second parameter is the normalized speed of slow writes. The third parameter controls the aggressiveness of *Bank-Aware Mellow Writes*: when the number of requests to the corresponding bank in the write queue is less than *bank_aware_threshold*, we issue the current write request as a slow write. A higher *bank_aware_threshold* usually results in longer NVM lifetime but lower performance.

- **Eager Mellow Writes (*eager_writebacks*).** This technique eagerly writes back dirty data in the LLC to NVM memory when it is not busy. It has three parameters: *slow_cancellation*, *slow_latency* and *eager_threshold*. The first two parameters are the same with *Bank-Aware Mellow Writes*. The third parameter, *eager_threshold*, controls the aggressiveness of *Eager Mellow Writes*: If the highest N LRU stack positions of last level cache (LLC) contributes less than $\frac{1}{eager\_threshold}$ total hits in LLC, then we consider these N LRU stack positions to be useless and their corresponding LLC dirty entry can be eagerly written back. A higher *eager_threshold* usually corresponds to higher performance but shorter NVM lifetime.

- **Wear Quota (*wear_quota*).** This technique divides the execution into multiple small time slices and assigns a wear quota to each slice. If at the beginning of time slice the accumulated wear quota is reached, the whole coming time slice can only use the slowest writes (in our implementation, $4\times$) and write cancellation is enforced. The parameter used here is *wear_quota_target* (in years), which indicates the target lifetime of *Wear Quota* technique. A larger *wear_quota_target* enforces longer NVM lifetime, at the expense of a lower system performance.

4.3.2 Objective tradeoffs

Since NVM systems face multiple constraints (e.g., performance, lifetime and possibly energy [116] as in embedded systems and data centers), the optimization objectives are usually complex and user-defined. In our case, there are three optimization goals: first, the qualified configurations must guarantee a minimum lifetime; then, our goal is to achieve an IPC as high as possible; finally, among all the qualified configurations whose IPCs are within 95% of the maximum, we choose the one with the lowest system energy as the *optimal* configuration. Let P_i, T_i, E_i represent IPC, lifetime, and energy respectively of configuration i, $\forall i \in (0, N)$. Then the optimization problem can be formalized as follows:

$$\begin{aligned} \min_{i \in (0,N)} \quad & E_i \\ \text{subject to} \quad & T_i \geq t, \\ & P_i \geq 0.95 \times P^* \end{aligned}$$

Although we focus on optimizing IPC and energy efficiency under lifetime constraints in our case study, our framework could be generally applied to optimization problems under other constraints, eg. by switching the three metrics: IPC, lifetime and energy from constraints to objectives and vice versa. For example, in embedded systems, the objectives could be to enforce a constraint on energy, while maximizing performance and lifetime. In data centers, however, the objectives could be to guarantee a performance target, while maximizing lifetime and minimizing energy. Our learning framework is flexible with user-defined objective functions since the main challenge we want to solve is to model performance, lifetime and energy of all different configurations based on a small number of samples, yet at negligible cost at runtime.

4.3.3 Challenges

The optimization problem (i.e., choosing the *optimal* configuration) is trivial if we have the data for all configurations: $[P_i, T_i, E_i]$, $\forall i \in (0,N)$. However, in this section, we will show that it is impractical to do so for three reasons: huge configuration space, high sensitivity to applications and high sensitivity to user defined objectives.

Huge Configuration Space

As *mellow writes* covers a series of techniques and multiple tradeoffs, the whole configuration space is non-trivial, as is shown in Tables 4.2&4.3. To reduce its size, we add three constraints to remove impractical configurations:

- The *parameters* are used only when it is enabled by the selected techniques. For example, when *eager_writebacks* technique is not selected, *eager_threshold* is meaningless and thus not considered.
- The *slow_latency* must be larger than *fast_latency*.
- When *fast_cancellation* is true, we force *slow_cancellation* to be true. Prior work [32] shows that write cancellation for slow writes is more effective in improving performance compared with write cancellation for fast writes. Therefore, it does not make sense to have a technique which offers write cancellation for fast writes, but not for slow writes.

However, even with these constraints, the configuration space of *Mellow Writes* is still huge—there are 3,164 different configurations in total. Brute-force search of the whole configuration space is very expensive. In our experiments, in order to compare our framework with the ideal configuration, we simulate all the configurations for 10 applications , which consumed more than 300,000 computing hours. The high evaluation cost makes the selection of the optimal configuration at runtime quite challenging.

High Sensitivity to User Defined Objective

The choice of optimal configuration is highly affected by the user defined objective. For application `leslie3d`, we vary the minimal lifetime requirement from 4 years to 10 years. The results are shown in Table 4.4. Due to the experiment time constraint, we explored a limited configuration space without the usage of *Wear Quota* for this table, but the results clearly indicate that the optimal configuration varies with different user defined objectives.

High Sensitivity to Applications

Not only does the optimization objective affect the choice of optimal configuration, but also the currently executed application. Figure 4.1 shows the optimal under the default optimization objective (i.e., 8-year lifetime, an IPC within 95% of the maximum IPC while minimizing energy). For comparison purposes, we also have two representative configurations: *default*, which does not use any mellow writes techniques; and *baseline*, which is the static configuration used in prior work [32].

We can see that, the effectiveness of *baseline* configuration is far from ideal—for more than half of the applications, the performance of *baseline* is significantly lower than *ideal*. However, as is shown in Table 4.5, finding out the *ideal* configurations is certainly not an easy task. In fact, in all the ten evaluated applications, none of them share the same *ideal* configuration.

4.4 Memory Cocktail Therapy

To address the challenges and solve the constrained optimization problem as discussed in the case study, we propose *Memory Cocktail Therapy*, a general, learning-based framework to dynamically model IPC, lifetime and system energy of different combinations of techniques for each application. We first formalize the problem in Section 4.4.1 and then quantitatively analyze the problem complexity in Section 4.4.2. The high complexity of our problem space indicates that statistical modeling is necessary, therefore we investigate various learning algorithms and evaluate their performance and computational overhead in Section 4.4.3. Finally, in Section 4.4.4 we further improve the accuracy of selected learning algorithms based on the insights from both machine learning and computer architecture perspectives.

4.4.1 Problem Formalization

We formulate the configuration space and tradeoff space to have a well-defined interface (inputs and outputs) for learning and optimization procedures.

Configuration Space

We represent all configurations with 10-dimensional vectors:

$$x = \begin{bmatrix} bank\_aware \\ bank\_aware\_threshold \\ eager\_writebacks \\ eager\_threshold \\ wear\_quota \\ wear\_quota\_target \\ fast\_latency \\ slow\_latency \\ fast\_cancellation \\ slow\_cancellation \end{bmatrix} \tag{4.1}$$

For example, the following vector, $[1,1,1,32,0,0,1.5,3.0,0,1]^T$ represents a combination of techniques that uses bank-aware mellow writes with a threshold of 1, eager writebacks with an eager threshold of 32, fast and slow write latencies of 1.5x and 3.0x, and write cancellation only on slow writes.

Tradeoff space

We include three metrics in the objective space: IPC, lifetime and system energy, as discussed in 4.3.2. Therefore, we formulate the tradeoff space into 3-dimensional vectors:

$$y = \begin{bmatrix} IPC \\ lifetime \\ system\_energy \end{bmatrix} \tag{4.2}$$

Application	Top-3 most effective features
lbm	$-fast\_latency$, $+fast\_latency^2$, $+slow\_cancellation^2$
leslie3d	$+slow\_cancellation^2$, $-eager\_writebacks * slow\_cancellation$, $+eager\_writebacks * fast\_latency$
GemsFDTD	$+slow\_cancellation^2$, $-slow\_latency * slow\_cancellation$, $+slow\_latency$
stream	$-fast\_latency$, $+slow\_cancellation^2$, $-eager\_writebacks * fast\_latency$

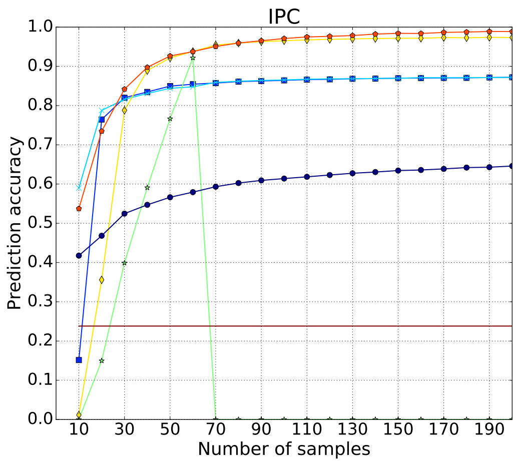
Table 4.6: Most effective quadratic features in different applications

4.4.2 Quantitative analysis of the problem space

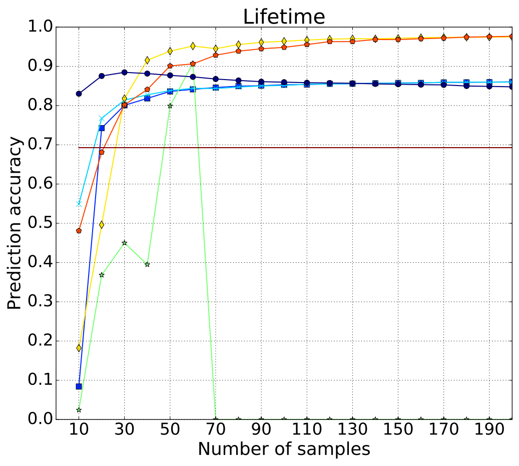
To demonstrate the problem complexity, we quantify the impacts of different input parameters and their correlation. We fit the training data to a quadratic regression model with lasso regularization, which show high prediction accuracy in later experiments. More details about the model will be introduced in Section 4.4.3. In this model, the input parameters are extended to quadratic features including both single knobs and knob pairs. The post-training weights of these features indicate their effectiveness and impacts on the outputs.

Then, we rank the top-3 most effective features for different applications, as shown in Table 4.6. From the effectiveness ranking results, we can see that: 1) some of these top-ranked features are knob pairs, which indicates high correlation between input parameters and the importance of their correlation. Thus, it is necessary to model the joint contribution of these parameters to the outputs. 2) different applications have entirely different top-ranked knobs/knob-pairs, so it is difficult to determine the effectiveness order statically. 3) single knobs can have nonlinear impacts on the outputs (e.g. *fast_latency* on *lbm*). Note also, we have a minimum lifetime constraint in the optimization problem. As a result, it is difficult to determine the best value for each knob to satisfy the lifetime constraint while maximizing performance and energy efficiency without statistical modeling.

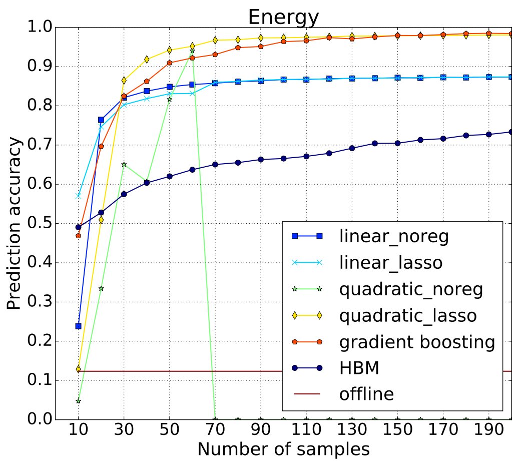
Therefore, we exploit machine learning techniques to model the relationship between configurations and the outputs, as discussed in the next section.



(a) IPC



(b) Lifetime



(c) Energy

Figure 4.2: Convergence rates and prediction accuracies using different models

4.4.3 Learning performance, lifetime and energy

To model performance, lifetime and energy, we investigate three learning algorithms: 1) *Regression*, 2) *Boosting algorithms*, 3) *Hierarchical Bayesian models*.

Regression models the functional relationship between configuration vectors and objective vectors, based on a small set of sample configurations. One drawback of regression is overfitting, which can happen when the model has many parameters (e.g., higher-order polynomials). Regularization reduces model complexity to avoid overfitting. Lasso, the least absolute shrinkage and selection operator, is a common regularization technique [117]. We will show that by using lasso, we can speed up the convergence of regression-based predictors (in Figure 4.2), as well as guide the feature selection for more informative runtime sampling (in Figure 4.4).

Boosting algorithms are a class of ensemble learning methods combining multiple weak learners into a strong learner [118]. Gradient boosting is a state-of-art boosting algorithm for learning regression models [119, 120]. It is also one of the most accurate methods for our data sets as shown later.

Hierarchical Bayesian models do not learn the functional relationship between inputs and outputs, but assume that some latent variables are shared between different applications and learning their posterior distributions allows predictions for the current application [121]. Rather than overfit, this model uses only similar applications to predict new application behavior; however, accuracy requires that the training set has sufficient breadth to find known applications that correlate with the new application.

Model selection

Table 4.7 and Figure 4.2 compare these models in terms of (1) requirements for online or offline data, (2) computation overhead (in microseconds), (3) convergence rates (in samples), and (4) prediction accuracies. We use *coefficient of determination* as our accuracy metric:

$$acc = \max\left(0, \left(1 - \frac{\left\|(Y' - Y)\right\|_2^2}{\left\|(Y - \overline{Y})\right\|_2^2}\right)\right) \tag{4.3}$$

where Y' represents the prediction, Y represents the true data and $\overline{Y}$ represents the mean of the true data. The coefficient of determination measures the proportion of the variance in dependent variables that can be predicted by independent variables; a commonly used metric to evaluate prediction accuracy [122]. The computation overhead is tested on a 12-core Intel i7 processor with 64 GB memory.

The *offline* predictor averages data from training applications to predict the current applica-

Predictors	Need offline data?	Need online data?	Computation overhead
offline	Yes	No	0 ms
linear model, no regularization	No	Yes	1 ms
linear model, lasso regularization	No	Yes	1 ms
quadratic model, no regularization	No	Yes	3 ms
quadratic model, lasso regularization	No	Yes	8 ms
gradient boosting	No	Yes	112 ms
hierarchical Bayesian model	Yes	Yes	8,000 ms

Table 4.7: Comparison of different models

tion. There is no runtime overhead, but prediction accuracy is low. The *Hierarchical Bayesian model* has high prediction accuracy and fast convergence rate on lifetime because of the high correlation among benchmark applications. However, the IPC and energy predictions are not accurate because the data magnitudes vary significantly among different applications. The biggest issue with this model, however, is that it takes 8,000 ms to produce a prediction. We will not focus on this predictor in this work. However, for future work, it is possible to design specialized hardware for the model to mitigate the runtime overhead. Among online models, *gradient boosting* and *quadratic regression with lasso regularization* both achieve high prediction accuracies on all three objectives and have reasonable runtime cost. *Quadratic model without lasso regularization* has difficulty converging before 200 samples. The reason is that the input vectors are expanded from 10 dimensions to 65 dimensions in the quadratic model, including square terms, cross terms and linear terms. Without regularization, the model is prone to overfitting given small sample size. *Linear models* are not as accurate as quadratic models, which indicates that the underlying features have complicated, interdependent relationships to the targets.

According to Table 4.7 and Figure 4.2, we choose gradient boosting and quadratic regression with lasso in our final experiments because they achieve high prediction accuracies with low computation overhead.

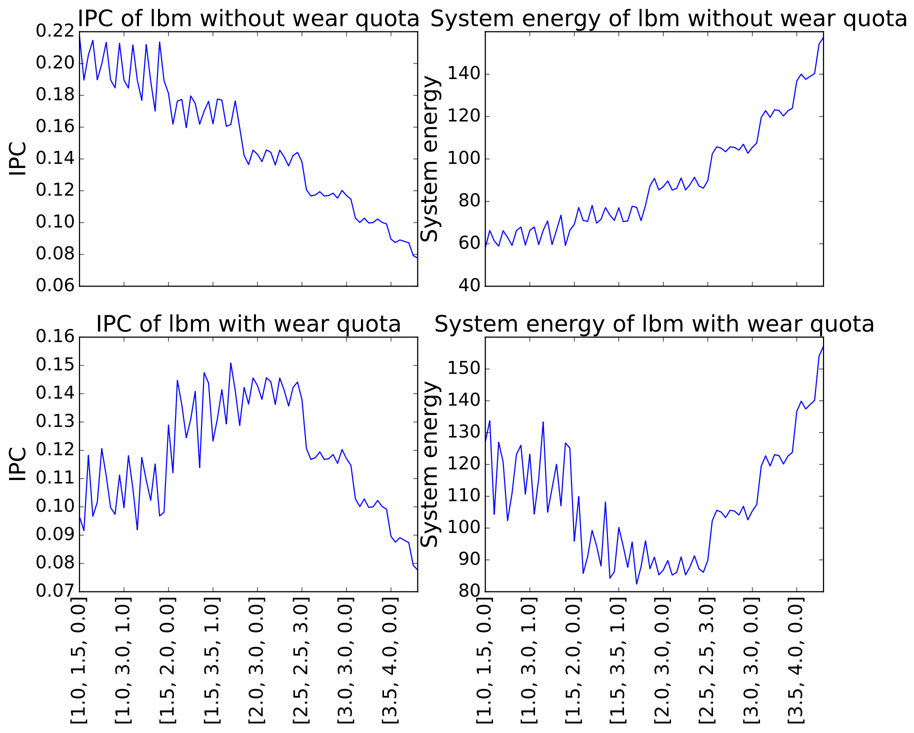


Figure 4.3: Including or excluding wear quota in the configuration space. We take benchmark *lbm* from SPEC CPU 2006 as an example and choose 77 sample configurations using feature-based sampling. The xlabels are in the format of: [*fast_rate*, *slow_rate*, *write_cancellation* (on slow and/or fast writes)]. From left to right, the configurations have increasing write latencies. Including wear quota adds more complexity to the modeling and degrades prediction accuracies by $2\% \sim 6\%$.

4.4.4 Improving Prediction Accuracies

Rather than applying existing learning techniques as black boxes, we can make small modifications to improve their behavior on the particular problem of optimizing non-volatile memory configurations. We find that prediction accuracy is improved using the following techniques:

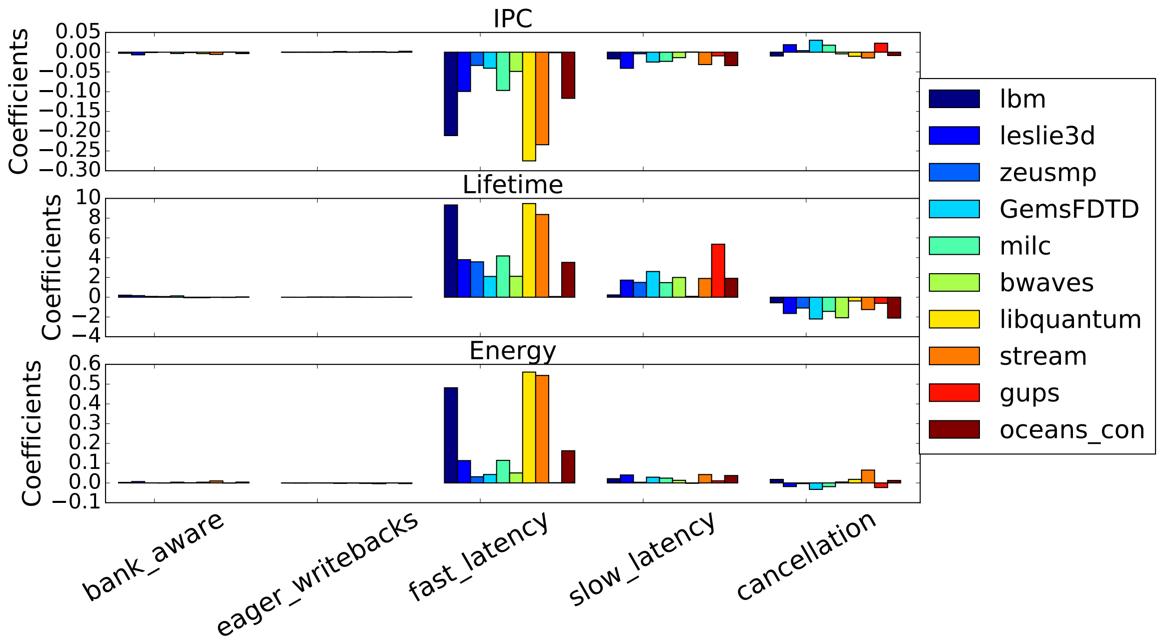
Normalization avoids extreme coefficient values for different parameters. We normalize all the data to the baseline configuration's measured behavior. This technique improves both the prediction accuracy and convergence rate; however, the predictor now only learns how different a configuration is from the baseline. Thus, we periodically run the baseline configuration and multiply the prediction data by the baseline data. This method accommodates small system phase changes and avoids oscillation by using absolute values.

Including or excluding wear quota. We observe that *wear quota* is a technique that makes significant impact on the three objectives. Wear quota is a technique to guarantee a minimum lifetime[32]. It does so by ensuring all memory writes are slow when the fast-write quota for a period is exceeded. Figure 4.3 illustrates the impacts of wear quota. Compared with the data not using wear quota, the data using wear quota exhibit higher complexity. Their performance and energy efficiency is hard to predict when the write latencies are either very low or very high. When write latencies are very low, wear quota is triggered, resulting in slow writes. When write latencies are very high, the memory system is intrinsically slow and its performance and energy efficiency is bad. We observe $2\% \sim 6\%$ degradation in prediction performance when including data using wear quota. Therefore, we exclude wear quota from our configuration space for prediction. However, we use it as a *fixup* technique later to guarantee the lifetime target for configurations whose lifetime were overestimated during prediction.

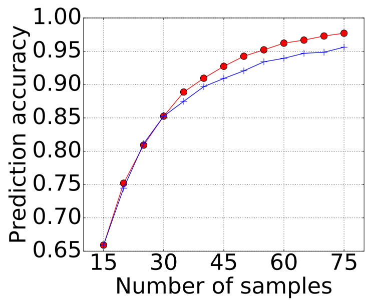
Feature selection. In the previous discussion, we use random sampling to compare different predictors. However, we find that the prediction accuracy increases if we sample based on prior knowledge about what features are most important in the configuration space.

To find important features, we first manually cluster the 8 features (excluding wear quota) based on domain knowledge. For example, *bank_aware* and *bank_aware_threshold* are merged into one variable, *bank_aware*, that has 5 levels from 0 to 4. Following this approach, we compress the original 8 features into 5 features: *bank_aware*, *eager_writebacks*, *fast_latency*, *slow_latency* and *cancellation*. This manual compression further aids dimensionality reduction. The configuration space, however, requires both the usage and the aggressiveness parameter for each technique.

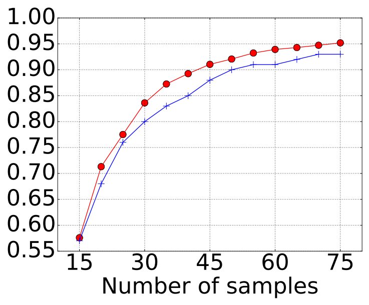
Then we use lasso regularization to identify important features. Figure 4.4a shows the coefficients of lasso regression with a linear model. The coefficients of *bank_aware* and *ea-*



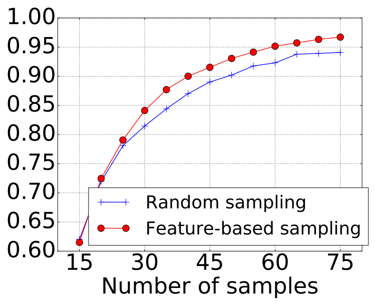
(a) Coefficients of lasso regression with linear model. Only important input features have nonzero coefficients.



(b) IPC



(c) Lifetime



(d) Energy

Figure 4.4: Feature-based sampling vs. random sampling on gradient boosting.

ger_writebacks are near zero for all objectives of all applications. The three important features are thus: *fast_latency*, *slow_latency*, and *cancellation*. The same features in square terms and cross terms are also the important features in the quadratic model with lasso regularization.

Based on the selected features, we obtain 77 samples by uniformly sampling from the three primary features and randomly sampling from the left. The feature-based sampling leads to higher prediction accuracies for all other configurations. For example, the performance of gradient boosting increases by about 3% on average for all objectives.

Additionally, the three primary features indicate that our framework is not limited to the techniques in this chapter. It could be generally applied to architectural techniques in NVMs that involve these three features [109, 112, 113].

4.5 Implementation

In this section, we discuss how to implement the learning-based framework in practical NVM systems. There are several challenges in the implementation. First, in real memory systems, there are various access patterns along time and among different applications. Meanwhile, the performance of architectural techniques varies significantly for different access patterns as they exploit different tradeoffs. Thus, we assume that the sample configurations and the chosen optimal configuration should run on similar memory behaviors. Otherwise, the information we learned from the training is not valuable.Second, real memory systems usually exhibit fine-grained bursty behaviors. However, all the different sample configurations in our framework should run on similar memory workload. We should schedule these samples carefully to avoid discrepancies in their workload. Third, although we have selected predictors with accuracies higher than 90%, there still exist mispredictions. Nevertheless, the final optimal configuration should still satisfy the hard constraint in the user-defined objective function, eg. the minimum lifetime target. Finally, we should guarantee that after optimization, our system will not perform worse than the baseline.

To address all these issues, we exploit a series of techniques, including phase detection, fine-grained runtime sampling, wear-quota fixup and periodic health checking. Figure 4.5 illustrates the workflow of our framework. And we discuss each technique in detail in the following subsections.

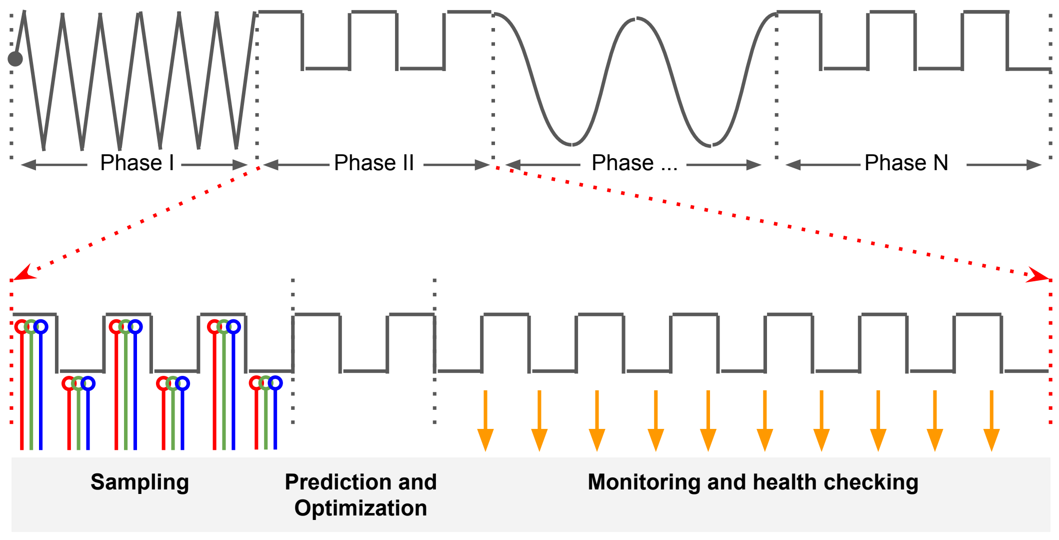


Figure 4.5: System implementation

4.5.1 Phase detection

Phase detection methodologies have been widely investigated in system area [123, 124, 121, 125, 126, 127]. Since memory systems exhibit frequent, fine-grained phase changes, our goal is to implement a lightweight phase detector that recognizes only dramatic changes in memory behaviors. Our system could tolerate minor phase changes by normalization, as discussed in Section 4.4.4, and fine-grained runtime sampling, as discussed in Section 4.5.2. Therefore, we adapt the phase detection methodology in [121] and use Student's t-test to emphasize the detection of dramatic phases. Our phase detection algorithm is as follows:

- Use performance counters to monitor the memory workload (including both read requests and write requests) for every I instructions.
- Keep a history record of the memory workload during the past $1000 * I$ instructions.
- Perform the two-sided Student's t-test for the last $100 * I$ instructions and the past $1000 * I$ instructions based on the mean values and standard deviations of their memory workload.

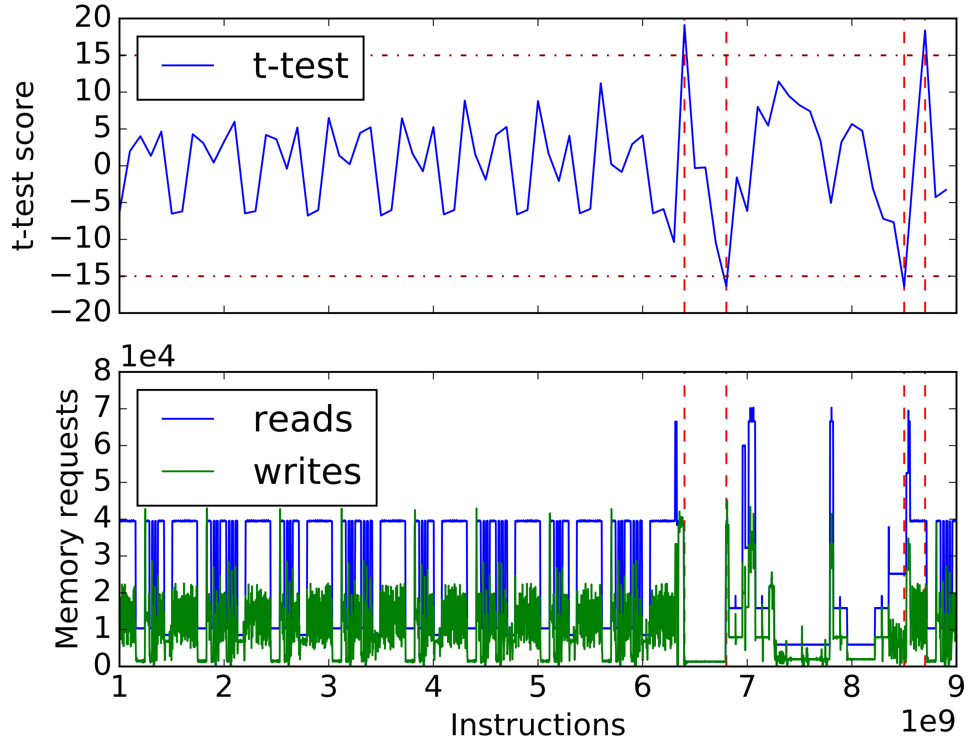


Figure 4.6: Phase detection. (I is 1,000,000 instructions and $score_{threshold}$ is 15.) Recognized phases in memory behaviors are marked with red vertical lines, which correspond with the changes in t-test scores.

The t-test score indicates the confidence in rejecting the hypothesis that the mean values of the memory workload in the two testing windows are the same. The higher the t-test score is, the more confident we are to recognize a new phase with respect to the memory behaviors.

- When the t-test score exceeds a threshold, $score_{threshold}$, we recognize a new phase. Then we clear off the counters and restart.

We demonstrate the phase detection result of *ocean* in Figure 4.6. Our phase detector can accurately capture coarse-grained phases, while tolerate frequent, fine-grained phases.

Note that, higher accuracy might be achieved by providing more program information to the phase detector. This can be done through either hardware modification (e.g., passing PC with memory requests) or application/compiler modification (e.g., generating compiler hints [124, 126, 125, 127]). In this work, the recognized phases shown in Figure 4.6 are sufficient to guide the learning procedures. Meanwhile, our phase detection scheme only relies on the statistics provided by existing performance counters.

4.5.2 Runtime sampling

Memory bursty behaviors are common in memory intensive applications (eg. *lbm*, *libquantum*, *milc*, etc.) [128]. If we evenly divide the sampling period into N slices for N samples, then some samples may fall in bursty periods while others fall in idle periods. To accommodate these fine-grained memory patterns, we propose fine-grained sampling:

- Assume the total sampling period covers T instructions.
- Run each sample configuration for a small sampling unit, which covers t instructions.
- Loop over all samples for $\frac{T}{N*t}$ times.
- Accumulate statistics of each sample configuration during $\frac{T}{N*t}$ sampling units.

We observe that the magnitude of memory burst length is at least 10 million instructions in our benchmarks. Hence, all the sample configurations could be scheduled within each memory burst if we use a small sampling unit, eg. 100 kilo instructions. Note also, the configuration of sampling unit length and the number of iterations can be guided by the mean memory workload in the current phase, as discussed in Sec 4.5.1. If the mean value is low, we can use large sampling units so that each sample configuration can make effects with enough memory requests. Otherwise, we can use small units and repeat many times so that all configurations can evenly experience all memory patterns.

The fine-grained sampling methodology is lightweight so that it incurs negligible overhead at runtime. With cyclic-fine-grained sampling, each sample will cover a wide set of memory behaviors. And all different samples are exercised on similar memory behaviors. Using the fine-grained sampling methodology together with the phase detector, our system can accommodate both fine-grained and coarse-grained phases in the memory system.

4.5.3 Prediction and optimization

After the sampling period, we collect the IPC, lifetime and energy data of the sample configurations. Then we apply the learning algorithms to predict the three objectives of all other configurations. After the prediction, we choose the optimal configuration based on the used-defined objective function, as defined in Section 4.3.2.

However, the optimization based on prediction data may have errors. For example, the chosen optimal configuration might be overestimated in lifetime while its real lifetime cannot satisfy the minimum target. To correct such prediction errors, we add *Wear Quota* to the chosen optimal configuration and set the *Wear Quota* target to the minimum lifetime target. Then the minimum lifetime could be guaranteed. If the chosen optimal configuration could meet the lifetime requirement itself, adding *Wear Quota* incurs negligible overhead according to the prior work. For prediction errors in IPC and energy, we account that in our final optimization results and compare them with the baseline and ideal policy.

4.5.4 Monitoring and health checking

After choosing the optimal configuration, we can launch the optimal configuration in the memory system. However, the chosen optimal configuration might be suboptimal in practice due to prediction errors. Hence, we monitor the performance of the memory system and periodically switch back to the baseline configuration for health checking. During this stage, we can perform several health checks of the current system. First, we can obtain the memory workload statistics for the phase detection. Second, we can check minor phase changes by just

Table 4.8: Processor Simulation Parameters

Freq.	2GHz
Core	Alpha ISA, single-core, OoO, 64-byte cacheline, 8-issue
L1$	split 32KB I/D-caches, 4-way, 2-cycle hit latency, 8-MSHR
L2$	256KB per core, 8-way, 12-cycle hit latency, 12-MSHR
L3$(LLC)	2MB, 16-way, 35-cycle hit latency, 32-MSHR

Table 4.9: Main Memory System Simulation Parameters

Basics	400 MHz, 4GB, 64-bit bus width, using ReRAM, assume using effective wear-leveling scheme (e.g., Start-Gap [30]) in bank granularity which can achieve 95% average lifetime, write-through (writes bypass row buffers), 1KB row buffer, open page policy, tFAW=50ns
# of Banks	16
# of Rows	8192 per bank
# of Cols	512 per row
Read Queue	64 entries, highest priority
Write Queue Eager Mellow Write Queue	64 entries, middle-high priority write drain threshold: 32 (low), 64 (high) 32 entries per channel, lowest priority, no write drain, slow writes
tRCD	48 cycles (120 ns)
tWP (WR pulse lat.)	$60 * wr\_ratio$ cycles ($150 * wr\_ratio$ ns).
tCAS	1 cycle (2.5 ns)
endurance	$8 * 10^6 * wr\_ratio^2$ writes;

monitoring the statistics of the baseline configuration. With the normalization technique, periodic statistics of the baseline configurations can help adapt to minor phase changes and avoid system oscillations. Third, if we find that the performance of the chosen configuration is worse than the baseline configuration, we can switch to the baseline configuration so that our system will never be worse than the baseline system.

4.6 Results

4.6.1 Experiment setup

We use gem5[12] and NVMain[129], which is a timing-accurate simulator for non-volatile memories. Table 4.8 and Table 4.9 respectively report the detailed parameters of processor and

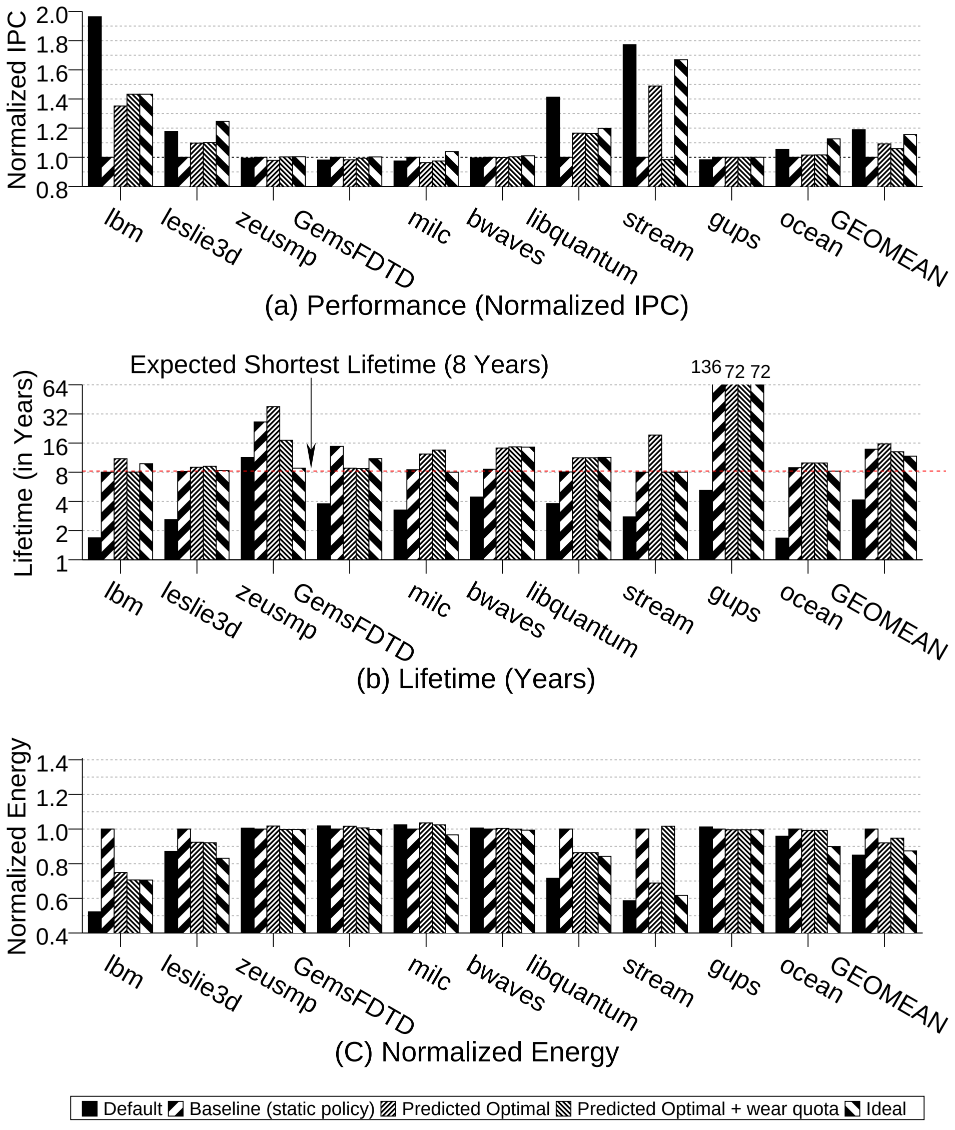


Figure 4.7: Compare *Memory Cocktail Therapy* with baseline systems with respect to IPC, lifetime and system energy.

resistive memory based main memory. We assume the memory system uses an efficient bank-level wear-leveling technique. Without loss of generality, we use ReRAM[130] for the simulated NVM-based main memory. According to recent represented commercial products[131], we model the baseline write latency to be 150ns and its endurance as 8×10^6. We also model the *Write Latency VS. Endurance* trade-off [32] in our simulation framework: the write latency can be extended to $150 * wr\_ratio$ ns, as a result, the write endurance can be improved quadrat-

	bank _aware	*bank _aware _threshold*	*eager _writebacks*	*eager _threshold*	*wear _quota*	*wear _quota _target*	*fast _latency*	*slow _latency*	*fast _cancel-lation*	*slow _cancel-lation*
static	True	1	True	32	True	8	1.0	3.0	False	True
lbm	True	1	True	32	True	8	2.0	2.5	False	False
libquantum	True	3	True	32	True	8	1.5	3.0	False	True
stream	False	N/A	True	32	True	8	1.5	2.5	False	False
ocean	True	2	True	4	True	8	1.5	3.5	False	True
bwaves	True	4	False	N/A	True	8	1.0	1.5	False	True

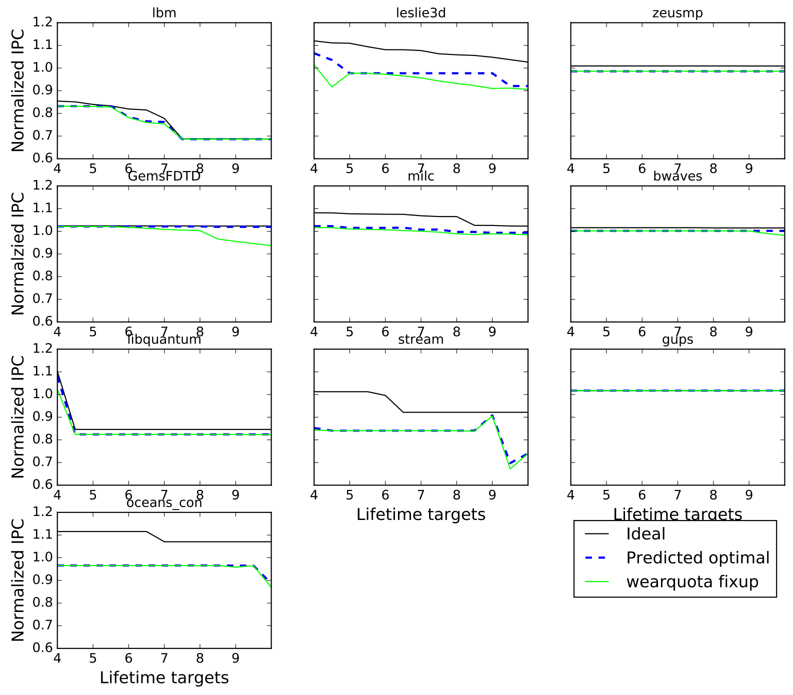
Table 4.10: Optimal configurations for different applications selected by MCT with gradient boosting.

ically to $8 * 10^6 * wr\_ratio^2$. For processor and memory systems, we respectively use MCPAT [132] and NVSim [133] to model their energy consumption.

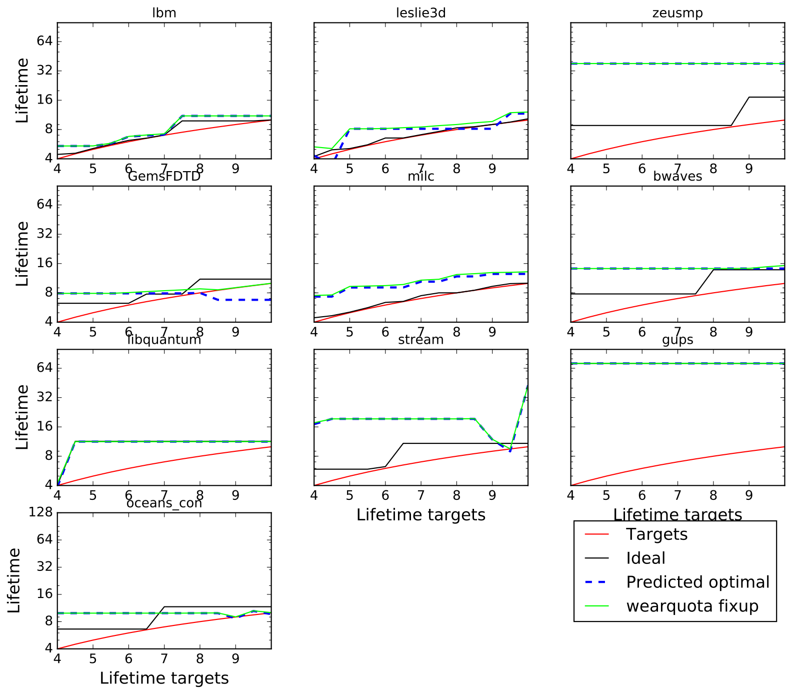
We run 7 memory-intensive workloads (*lbm, leslie3d, zeusmp, GemsFDTD, milc, bwaves* and *libquantum*) from SPEC CPU2006, *ocean* from SPLASH-2, and 2 extra microbenchmarks (*gups* and *stream*) which separately provides random and stream memory access pattern. Each benchmark is warmed up for 6 billion instructions and simulated in detail for another 2 billion instructions. To calculate lifetime, we assume the system will cyclically execute the current workload until the main memory wears out, and this total execution time is the memory lifetime for the workload.

4.6.2 Evaluation of memory cocktail therapy

In this section, we evaluate *MCT* with respect to our three objectives: *IPC*, *lifetime* and *system energy*. We compare *MCT* with three baseline systems: 1) *default*, which does not use mellow writes techniques, 2) *best static policy* chosen from prior work, which uses bank aware mellow writes and eager writebacks with slow write cancellation and wear quota. 3) *ideal policy*, which is selected by a brute-force search through the whole configuration space. We demonstrate *MCT* using gradient boosting and quadratic lasso as the learning models without using wear quota in the learning process. After the prediction, *MCT* uses wear quota as a fixup technique for the predicted optimal configuration to guarantee a minimum lifetime target.

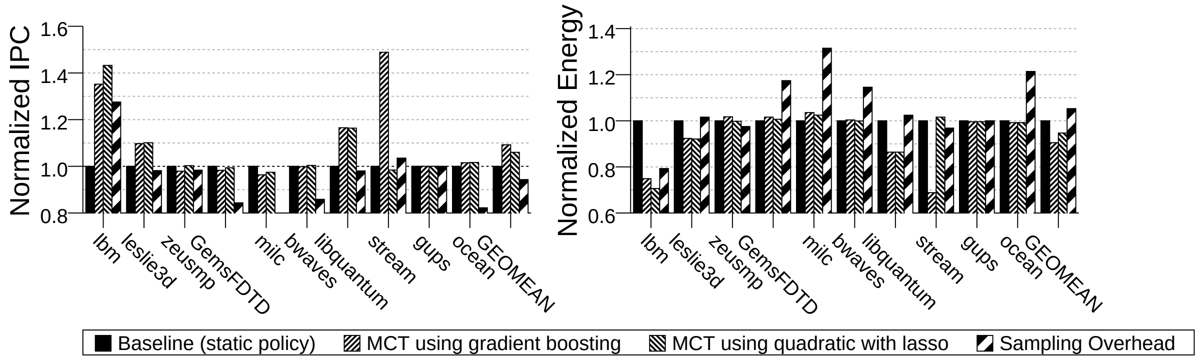


(a) IPC

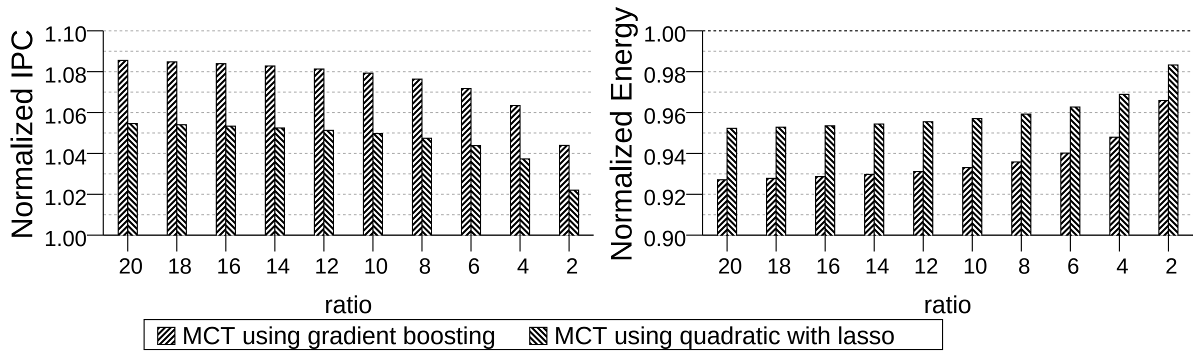


(b) Lifetime

Figure 4.8: Sensivity to different lifetime targets



(a) Gains vs. losses



(b) Extrapolate gains and losses

Figure 4.9: Sampling overhead

Better tradeoffs between IPC, lifetime and system energy

Figure 4.7 shows the comparison of *MCT* with other baseline systems. We assume that the lifetime requirement is 8 years here. And we normalize the IPC and energy data by the *best static policy*. The *default* system only uses fast writes so it achieves high IPC and low energy consumption. However, it cannot satisfy the minimum lifetime requirement for most of our benchmarks (except *zeusmp*). Both *MCT* and the *best static policy* can guarantee the minimum lifetime requirement, while *MCT* using gradient boosting achieves 9.24% higher IPC and 7.95% lower energy consumption on average for all the benchmarks. Particularly, *MCT* achieves significantly better tradeoffs when the *best static policy* is far behind the *ideal policy*, as in *lbm*, *leslie3d*, *libquantum*, and *stream*. For other benchmarks that the static policy is already close to the ideal policy, *MCT* performs similarly to the baseline with negligible degradation in performance and energy efficiency. We list the optimal configurations selected by MCT in Table 4.10. Small variations in the configuration can result in significant improvement over the static policy. For example, in *lbm*, the only differences between the MCT-optimal configuration and the static configuration are in *fast_latency*, *slow_latency*, and *slow_cancellation*. However, the performance improvement is up to 35%. The main reason is that, the high *slow_latency* together with *slow_cancellation* in the static policy causes many rewrites, which hurts lifetime. Eventually, more wear quota slow writes are enforced in order to guarantee the minimum lifetime target of 8 years.

Compared with the *ideal policy*, *MCT* using gradient boosting achieves 94.49% of the maximum performance with only 5.3% more energy consumption. *MCT* using quadratic with lasso achieves 6% performance gains and 5.3% energy savings compared with the static policy. And it achieves 91.69% of the ideal performance with 8.3% more energy. In particular, it performs well on most of the benchmarks except *stream*. This indicates that quadratic model may not work well for every application depending on the intrinsic memory characteristics while gradient boosting is more general as it includes a variety of weak predictors.

Sensitivity to different lifetime targets

Figure 4.8 shows the results of *MCT* with different lifetime targets. We only show MCT using gradient boosting here. In our experiments, the lifetime targets range from 4 years to 10 years. In general, when we have higher lifetime targets, the chosen optimal configuration has lower performance and higher energy consumption, and vice versa. *MCT* can generally capture this trend. However, there are discontinuities in the predictions, as well as the ideal policies for different lifetime targets. One reason is that, the set of configurations we experiment on is still far from complete compared with the oracle, although it already contains 3,164 configurations and needs more than 300,000 computing hours to simulate for all benchmarks. Nevertheless, MCT still manages to select a better configuration than the baseline configuration as in *lbm, stream, lelsie3d*, etc. while saving large amount of evaluation time for each configuration. When the chosen optimal configuration has overestimated lifetime (eg. in *GemsFDTD* when the lifetime target is beyond 8 years), the wearquota fixup technique works as the last resort to guarantee that the minimum lifetime requirement will still be satisfied.

Reduced sampling and learning complexity by excluding wear quota in the learning process

Although for some benchmarks (eg. *lbm* and *stream* as shown in Table 4.5), including wear quota in the learning process leads to better choices, there are two practical problems with wear quota prediction: 1) The prediction accuracies including wear quota degrades by 2% $\sim$ 6% for all applications, as discussed in Section 4.4.4. 2) Wear quota is a technique that depends on aggregate memory behaviors in a long period. However, we need short sampling period to mitigate runtime sampling overhead and fine sampling granularity to tolerate memory bursty behaviors. Including wear quota in the prediction space adds more challenges in practical sampling methodologies. For example, using the same fine-grained sampling methodology, the chosen optimal configuration for *lbm* including wear quota only achieves 70% of the performance by excluding wear quota, while consuming 50% more energy. And we observe similar phenomenon in *leslie3d*, which has 6% performance degradation with 13% more energy consumption when including wear quota in the prediction.

Sampling overhead

During the sampling period, MCT exercises different configurations in order to model IPC, lifetime and energy for all configurations. However, these sample configurations are generally not the optimal one. In this section, we evaluate the overhead caused by running suboptimal configurations and compare the gains in testing period to the losses in the sampling period. We refer the rest of the phase as the sampling period to testing period here.

Figure 4.9a shows the comparison. All the data are normalized by the static policy. On average, the aggregate IPC of sample configurations is 94.32% of the baseline, while MCT using gradient boosting achieves 1.09x IPC of the baseline. Similarly, the aggregate energy consumption of sample configurations is 1.05x of the baseline, while the predicted optimal configuration only consumes 92.05% energy of the baseline.

Considering the overhead during the sampling period, the practical gains of MCT depend on the ratio between the testing period length and the sampling period length. As a proof of concept, in our experiments, the sampling period covers 1 billion instructions and the testing period covers 2 billion instructions. However, according to prior work on memory workload characterization of SPEC CPU 2006 benchmarks[128], there are only a few phases during the whole program execution. And each program has thousands of billions instructions. To have a better idea about the practical gains, we extrapolate the overhead and gains with different ratios between the testing period length and the sampling period length. If the testing period is α times of the sampling period, then the extrapolated IPC is:

$$IPC_{total} = (IPC_{sampling} + \alpha * IPC_{testing})/(1+\alpha) \tag{4.4}$$

Figure 4.9b shows the total IPC and energy consumption by extrapolating gains and losses based on Equation (4.4). For example, if the testing period is 10x of the sampling period, which is a reasonable case from the characterization results[128], then *MCT* using gradient boosting can still achieve 7.93% performance gains and 6.7% energy savings compared with the static policy.

Extension to multi-program workloads

We also investigate the effectiveness of MCT in a multi-core architecture. The architecture has 4 cores, independent L1/L2 cache for each core, a shared 8MB L3 cache and an 8GB,

Table 4.11: Multi-program workloads

mix1	lbm, libquantum, stream, ocean
mix2	leslie3d, bwaves, stream, ocean
mix3	GemsFDTD, milc, zeusmp, bwaves
mix4	lbm, leslie3d, zeusmp, GemsFDTD
mix5	GemsFDTD, milc, bwaves, libquantum
mix6	libquantum, bwaves, stream, ocean

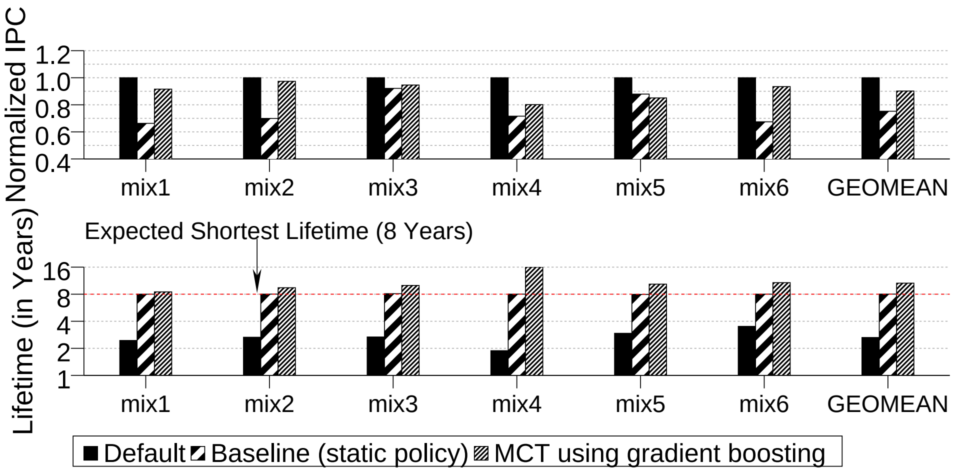


Figure 4.10: *Memory Cocktail Therapy* in multi-core environments.

32-bank resistive main memory. We randomly pick 4 benchmarks and execute them concurrently, and then evaluate the performance (in normalized geometric mean IPC) and lifetime (in years). The multi-program workloads that we experimented on are listed in Table 4.11. Our results demonstrate that MCT also performs well in multi-program situations, as shown in Figure 4.10. Compared to the static policy, MCT achieves around 20% (geometric mean) performance benefits and also satisfies the 8-year lifetime requirement. We only compare MCT with the static policy for single-core architectures because exploring the design space in multi-core architectures for comparison is computationally intractable. Meanwhile, MCT can automatically find a good configuration for the current multi-program workload by only exercising a small set of sample configurations at runtime.

We found that, multi-core workloads tend to smooth phase behavior out as soon as they have a least a few programs. Similar results have also been reported in the literature [134].

That is why MCT also performs well in multi-core environments. The only dramatic effect, however, is when one or more programs start/exit. In such situations, our phase detection scheme is still necessary to provide rapid adaptation to this large shift in memory behavior. The effectiveness and fairness of MCT can be further improved by exploiting multi-program specific characteristics (e.g., by utilizing the schemes similar to [135, 136]) and we leave it as our future work.

4.7 Related Work

Machine learning approaches have attracted more interest recently to assist the architecture design as systems and architectures are becoming more complicated. These techniques have been used for automatic resource allocation [104, 106], branch prediction/LLC reuse-distance prediction [137, 138, 139], performance modeling [140, 141, 142], etc. For example, ensembles of Artificial Neural Networks (ANNs) were exploited to coordinate the allocation of multiple interacting resources to different applications and thus optimize the system-level performance [104]. Reinforcement learning (RL) was used to adapt the memory scheduling policies to changing workload and maximize the memory utilization [106]. Markov Decision Process (MDP) was used to model how the RL-based memory controller interacted with the rest of system.

Particularly, machine learning approaches can provide guidance on how to determine the optimal architectural configuration for specific applications and targets. LEO [121] exploited a hierarchical Bayesian model to learn the system behaviors of various processor and memory configurations and then minimize energy under performance constraints. JouleGuard [143] employed a combination of machine learning (eg. multi-armed bandits) and control theoretic techniques (eg. PI controller) to provide energy guarantees while maximizing accuracy. CASH [105] also exploited a combination of control theory and machine learning techniques to learn fine-grained configurations of multi-core architectures in IaaS Clouds to provide near-optimal cost savings for different QoS targets.

Our proposal extends the application of machine learning approaches to emerging memory technologies, which introduced a new constraint in the optimization space: lifetime. The lifetime constraint increases optimization complexity and leads to significantly different optimal configurations according to our results. Furthermore, compared with prior machine learning approaches, our framework is very lightweight: it does not require offline training, hardware modification or OS coordination and it incurs negligible runtime overhead. Also, our machine learning methodology is very straightforward and can be easily generalized to solve other architecture problems.

4.8 Chapter Summary

This chapter introduces *Memory Cocktail Therapy* (MCT), a general learning-based framework that manages to optimize combined techniques which utilize multiple dynamic trade-offs in NVM. With minimal performance overhead and no hardware modification, MCT manages to find the near-optimal configuration for the current application under a user-defined objective out of thousands of candidate configurations. MCT reduces the dimensionality of the configuration space with lasso regularization and selects three primary features: *fast_latency*, *slow_latency* and *write cancellation*. These are general features in NVM techniques so that our framework can also be applied to the optimization of other NVM techniques. Moreover, MCT manages to accommodate both fine-grained and coarse-grained phases by phase detection and cyclic-fine-grained runtime sampling.

We implement MCT using both gradient boosting and quadratic regression with lasso. We demonstrate that MCT using gradient boosting achieves better optimization results on average for all applications. For example, to guarantee an 8-year lifetime, achieve an IPC that is within 95% of the maximum, and minimize energy, MCT using gradient boosting improves the performance by 9.24% and reduces energy by 7.95% compared to the best static configuration. Compared with the ideal configuration in each application, MCT achieves 94.49% of the performance with only 5.3% more energy consumption (geometric mean).

Chapter 5

Future Work

In this thesis, we explored the opportunities of exploiting emerging technologies to build novel architectures. We exploited multi-level PCM to provide efficient storage of large-scale hash tables. We built limited-use security architectures from NEMS switches using self-degradation measures. And we optimized NVM systems to automatically adapt to different applications and objectives to achieve the best tradeoff between performance, lifetime, and energy efficiency. In addition to these architectures, we discuss several future directions of using emerging technologies in the following sections.

5.1 Hardware support for datastructures

There are other data structures that can benefit from the variable storage in multi-level PCM except for hash tables. Here we talk about inverted index structures and prefix trees (trie) in large scale web search engines.

5.1.1 Scalable inverted index structures in multi-level PCM

Inverted index structures are commonly used in many domains for efficient queries. For example, in web search engines, the inverted index structure is used to index and search related documents based on the keywords. Each of the keywords is accessed as the index to retrieve a list of documents that contain the word. Due to the large number of web documents, the storage efficiency and query performance of the inverted index becomes critical to the performance of search engines. Although solutions such as data compression could improve the storage efficiency, they cannot improve the query performance at the same time.

As the basic organization and operations of inverted index structures are very similar to hash tables, they can also benefit from the multi-level PCM. First, the popularity of different words varies significantly, which results in asymmetrically linked document lists in the inverted index. As shown in herniated hash tables, multi-level PCM can accommodate variable lengths of lists by expanding the storage on demand in deeper levels of the same memory cells. And prefetching techniques can help hide the longer read latency of deeper list entries. Furthermore, the document lists in inverted index structures are usually sorted based on the access frequency of these documents. To maintain the sorted order of the document list in traditional storage is more expensive: it is necessary to rewrite the entire document list if the order changes. In multi-level PCM, however, keeping the sorted order is a free byproduct because the iterative write mechanism rewrites all the data on every data update. The sorted order will in turn improve the query performance by alleviating unnecessary data retrieval that are stored in very deep levels. As a result, we envision higher performance and density benefits for inverted index structures than general hash table structures.

5.1.2 Efficient storage of prefix trees

Prefix trees are a commonly used data structure for the efficient search of popular keywords and phrases. A typical application is the *Typeahead* service in web search engines: after a user typed in several characters of the keyword, the search engine automatically completes the keyword or prompts with several most possible keywords the user wants to search about. In the prefix tree, each path represents a valid keyword. Each tree node does not store a key value, but it stores references to all its children who share the same prefix (path to the parent node) but have different following characters. The auto-completion of keywords basically needs to scan the prefix tree and retrieve all the paths starting from the node with the prefix.

It is promising to exploit multi-level PCM to store all the children references of each node. First, although there are always 26 valid characters in English words, the number of popular suffixes varies largely for different prefixes. We can pack all the references to valid suffixes in each node in the same PCM cells. Multi-level PCM can compress the storage for each node and offer in-place expansion for nodes with variable numbers of children. Second, the access pattern of prefix trees is usually breath-first search starting from a specific node. As a result, a traversal of all the children references in each node is necessary. Although the data compression of each node in multi-level PCM may break the fast indexing feature in traditional array structures, the performance of breath-first search will not be compromised a lot, especially considering opportunities for prefetching. Third, the prefix tree will be rarely updated after the initialization, which can avoid expensive write operations in multi-level PCM. Above all, multi-level PCM can offer dense and non-volatile storage of these prefix trees, as well as fast queries on demand. It will be interesting to explore the opportunities to improve the quality of the *Typeahead* service with such hardware support.

5.2 Rate Limiting Security architectures

In *Lemonade from Lemons*, we built limited-use security architectures by harnessing the wearout phenomenon in NEMS switches: the failure probability of NEMS devices increases with more switching cycles, which imposes a usage upper bound on the entire system, so as to resist brute-force attacks. We proposed a series of architectural techniques and redundant encoding methods to differenciate legitimate users and attackers, by creating a steep boundary in their probabilities of gaining successful accesses to the device storage. However, there are inherent differences in their access patterns: legitimate users tend to consume the access quota evenly every day, but attackers will attempt to crack the device in a very short time. Therefore, in addition to the number of switching cycles, the frequency of switching operations can also be exploited to distinguish legitimate users and attackers.

Meanwhile, many NEMS switches are composed of materials with multiple electrical and mechanical functionalities, such as polymer matrix composites, which also exhibit the ability to automatically heal the premature facture and reduced durability [144, 145]. To enable the self-healing, however, the switching rate must be limited so that the devices can recover in between consecutive accesses. In opposite, the access pattern of brute-force attackers does not give enough time for the devices to heal premature issues.

The self-healing and rate-limiting features have brought more opportunities to enhance the security of the *Lemons* architectures. On the one hand, the self-healing feature can extend the lifetime of the NEMS network for legitimate users if the user consumes the access quota uniformly. On the other hand, the rate-limiting effect and potentially accelerated wearout can throttle brute-force attacks in a limited amount of time. The main challenge will be to model the self-healing behaviors of NEMS materials and characterize these materials with the required latency and energy for the self-healing. It will be interesting to analyze the minimum access-interval constraint to elongate the system-level lifetime.

5.3 Statistical Inference and Optimization of NVM Systems

5.3.1 Identifying program characteristics that interference with NVM techniques

From the motivational experiments in the *Memory Cocktail Therapy*, the optimal NVM techniques for different applications can be significantly different. An interesting question, however, is what program characteristics attribute to their preferences for these NVM techniques? For example, due to the tradeoff between the write latency and write endurance in phase change memories, various techniques attempt to schedule slow write operations in a best-effort manner, write back early and slowly while not degrading the system performance. As a result, program characteristics such as the type of data structures, stride access patterns, update frequencies, etc. may have significant impact on cache-block reuse distance, memory workload and read/write distribution, and consequently affect the best choice of NVM techniques.

Identifying correlated program characteristics can bring the following benefits to the current MCT framework. First, reduce the sample size. In the current framework, we use a general set of sample configurations (77 samples) learned from all the training applications. To exercise these sample configurations at runtime causes a performance overhead (the sampling period takes 1 billion instructions). Given the knowledge of specific program characteristics, we can rule out some sample configurations that we can predict with high confidence offline and then focus on exercising those architectural parameters that may have large variance. Second, learn from training applications without the runtime overhead. MCT did not use the hierarchical Bayesian model because it is too expensive to train at runtime. Actually, it attempts to learn the correlation of the target application with other training applications after collecting some runtime samples of the target application. However, if we can characterize these applications statically, we will not need the online training to learn the correlation between applications. Moreover, categorize applications based on their program characteristics and deploy them accordingly to heterogeneous NVM systems.

5.3.2 Approximate computing with NVM systems

In the current *Memory Cocktail Therapy* framework, we introduce three system metrics, performance, lifetime and energy efficiency. Meanwhile, as applications and data scale, more and more applications exhibit tolerance to a small amount of errors in outputs. Taking this new metric, accuracy, into account, we can explore more interesting tradeoffs in NVM systems. For example, instead of rewriting after a write cancellation, we can keep the partially-written value in the memory cells if it is close to the target value. Thus, we can avoid the cost at the system performance and lifetime during rewrite operations. Also, similar learning frameworks as MCT can help determine the best NVM techniques for different applications adaptively.

The challenging issue, however, is how to provide statistical guarantee on the accuracy target. The program characterization discussed in Section 5.3.1 can be helpful. The impact of approximating different program regions on the system performance, lifetime and accuracy will be different. Hence, the static analysis of the correlation can provide guidance on whether we should enable the approximation of write operations.

Chapter 6

Conclusion

In this thesis, we have presented three cases of novel architectures using emerging technologies and demonstrated the value of the statistical design methodology.

In *Herniated Hash Tables*, we provide hardware-level optimization to traditional datastructures, hashtables, by storing multiple hash collisions in multilevel PCM. The density benefits of multilevel PCM, however, come with the exponential latency overhead when reading deeper bits within each cell. To avoid the degradation of system performance, we propose three prefetching options and address mapping techniques. Experimental results show that Herniated Hash Tables can get up to 4.8x density benefits while achieving up 67% performance speedup over traditional chained hash tables on single-level PCM.

In *Lemonade from Lemons*, we build physically-enforced limited-use security architectures from NEMS switches. We characterize the wearout behaviors of NEMS switches with a probabilistic failure model: Weibull distribution. Then we design architectures that can physically limit attacks while accommodating legitimate usage. Three use cases are examined: a limited-use connection, a limited-use targeting system and one-time pads. We propose a family of architectural techniques to provide statistical guarantee on the minimum and maximum usage bounds given device variability. We perform extensive engineering space exploration for each case to discuss the tradeoffs between fabrication cost, area and energy. For example, redundant encoding can effectively reduce the area from exponential scaling to linear scaling with the increase of device usage bounds. Overall, we envision new opportunities for physically limiting vulnerability to attacks through careful engineering of intentional device wearout.

In *Memory Cocktail Therapy*, we exploit machine learning techniques to optimize the various NVM techniques that utilize multiple dynamic trade-offs for different applications. According to the experiments, MCT manages to find the near-optimal configuration for the current application under a user-defined objective. Also, compared to other adaptive systems, MCT incurs minimal performance overhead and no hardware modification. In particular, MCT selects three most important features from the thousands of configurations: fast latency, slow latency and write cancellation, which provide guidance on how to improve future NVM systems. Also, these features are general in NVM techniques so that our framework can also be applied to the optimization of other NVM techniques.

From these architectures, we learned the following benefits of statistical techniques. First, as architectures are becoming more complicated, the number of features in architectures increases significantly. However, these features may have different importance. It is very challenging to manually differentiate their importance without a large amount of comparison experiments. Second, the multi-input/multi-output relationship between the features and the architectural behaviors could be very complicated and counter-intuitive. Meanwhile, the statistical modeling can capture the relationship accurately at low cost. Third, if we can approximate the complex end-to-end relationship in programs with functional relationship, we will be able to reason about the performance bottlenecks and have more opportunities to accelerate and optimize the program at the hardware level. Finally, the potential benefits of using statistical techniques also include that, they can learn as they go: updating the feature weights and the model coefficients while seeing more data.

Bibliography

[1] M. K. Qureshi, M. M. Franceschini, L. Lastras-Monta, *et. al.*, *Improving read performance of phase change memories via write cancellation and write pausing*, in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–11, IEEE, 2010.

[2] F. Alibart, L. Gao, B. Hoskins, and D. B. Strukov, *High-precision tuning of state for memristive devices by adaptable variation-tolerant algorithm*, *CoRR* **abs/1110.1393** (2011).

[3] L. Zhang, D. Strukov, H. Saadeldeen, D. Fan, M. Zhang, and D. Franklin, *Spongedirectory: Flexible sparse directories utilizing multi-level memristors*, in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, pp. 61–74, 2014.

[4] B. Jacob and T. Mudge, *Virtual memory in contemporary microprocessors*, *Micro, IEEE* **18** (1998), no. 4 60–75.

[5] B. Jacob and T. Mudge, *Virtual memory: Issues of implementation*, *Computer* **31** (1998), no. 6 33–43.

[6] B. Jacob and T. Mudge, *Software-managed address translation*, in *High-Performance Computer Architecture, 1997., Third International Symposium on*, pp. 156–167, IEEE, 1997.

[7] B. L. Jacob and T. N. Mudge, *A look at several memory management units, tlb-refill mechanisms, and page table organizations*, in *ACM SIGOPS Operating Systems Review*, vol. 32, pp. 295–306, ACM, 1998.

[8] D. L. Weaver and T. Gremond, *The SPARC architecture manual.* PTR Prentice Hall Englewood Cliffs, NJ 07632, 1994.

[9] M. Talluri, M. D. Hill, and Y. A. Khalidi, *A new page table for 64-bit address spaces*, vol. 29. ACM, 1995.

[10] K. Diefendorff, R. Oehler, and R. Hochsprung, *Evolution of the powerpc architecture*, *IEEE Micro* (1994), no. 2 34–49.

[11] J. Huck and J. Hays, *Architectural support for translation table management in large address space machines*, in *ACM SIGARCH Computer Architecture News*, vol. 21, pp. 39–50, ACM, 1993.

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, *The gem5 simulator*, *SIGARCH Comput. Archit. News* **39** (2011), no. 2 1–7.

[13] A. Cortex, *a9 processor*, *URL: http://www. arm. com/products/processors/cortex-a/cortex-a9. php.[Accessed 6 January 2014]* (2011).

[14] M. Poremba, T. Zhang, and Y. Xie, *Nvmain 2.0: Architectural simulator to model (non-) volatile memory systems*, .

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, *The parsec benchmark suite: Characterization and architectural implications*, in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, ACM, 2008.

[16] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, *Morphable memory system: A robust architecture for exploiting multi-level phase change memories*, in *37th International Symposium on Computer Architecture*, pp. 153–162, 2010.

[17] R. Zhou and T. Li, *Leveraging phase change memory to achieve efficient virtual machine execution*, in *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 179–190, 2013.

[18] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, *Improving write operations in mlc phase change memory*, in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pp. 1–10, IEEE, 2012.

[19] M. Joshi, W. Zhang, and T. Li, *Mercury: A fast and energy-efficient multi-level cell based phase change memory system*, in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 345–356, IEEE, 2011.

[20] H. Saadeldeen, D. Franklin, G. Long, C. Hill, A. Browne, D. Strukov, T. Sherwood, and F. T. Chong, *Memristors for neural branch prediction: a case study in strict latency and write endurance challenges*, in *Proceedings of the ACM International Conference on Computing Frontiers*, pp. 26:1–26:10, 2013.

[21] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, *Approximate storage in solid-state memories*, in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 25–36, 2013.

[22] M. M. Michael, *High performance dynamic lock-free hash tables and list-based sets*, in *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 73–82, ACM, 2002.

[23] O. Shalev and N. Shavit, *Split-ordered lists: Lock-free extensible hash tables*, *Journal of the ACM (JACM)* **53** (2006), no. 3 379–405.

[24] J. Triplett, P. E. McKenney, and J. Walpole, *Resizable, scalable, concurrent hash tables via relativistic programming.*, in *USENIX Annual Technical Conference*, p. 11, 2011.

[25] N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun, *Transactional predication: high-performance concurrent sets and maps for stm*, in *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pp. 6–15, ACM, 2010.

[26] X. Li, D. G. Andersen, M. Kaminsky, and M. J. Freedman, *Algorithmic improvements for fast concurrent cuckoo hashing*, in *Proceedings of the Ninth European Conference on Computer Systems*, p. 27, ACM, 2014.

[27] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, *A large-scale study of flash memory failures in the field*, in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 177–190, ACM, 2015.

[28] J.-A. Carballo, W.-T. J. Chan, P. A. Gargini, A. Kahng, and S. Nath, *Itrs 2.0: Toward a re-framing of the semiconductor technology roadmap*, in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pp. 139–146, IEEE, 2014.

[29] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, *Free-p: Protecting non-volatile memory against both hard and soft errors*, in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 466–477, IEEE, 2011.

[30] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, *Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling*, in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 14–23, ACM, 2009.

[31] H. Saadeldeen, D. Franklin, G. Long, C. Hill, A. Browne, D. Strukov, T. Sherwood, and F. T. Chong, *Memristors for neural branch prediction: a case study in strict latency and write endurance challenges*, in *Proceedings of the ACM International Conference on Computing Frontiers*, p. 26, ACM, 2013.

[32] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, *Mellow writes: Extending lifetime in resistive memories through selective slow write backs*, in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 519–531, IEEE, 2016.

[33] O. Y. Loh and H. D. Espinosa, *Nanoelectromechanical contact switches*, *Nature nanotechnology* **7** (2012), no. 5 283–295.

[34] J. O. Lee, Y.-H. Song, M.-W. Kim, M.-H. Kang, J.-S. Oh, H.-H. Yang, and J.-B. Yoon, *A sub-1-volt nanoelectromechanical switching device*, *Nature nanotechnology* **8** (2013), no. 1 36–40.

[35] Z. Shi, H. Lu, L. Zhang, R. Yang, Y. Wang, D. Liu, H. Guo, D. Shi, H. Gao, E. Wang, *et. al.*, *Studies of graphene-based nanoelectromechanical switches*, *Nano Research* **5** (2012), no. 2 82–87.

[36] X. Feng, M. Matheny, C. A. Zorman, M. Mehregany, and M. Roukes, *Low voltage nanoelectromechanical switches based on silicon carbide nanowires*, *Nano letters* **10** (2010), no. 8 2891–2896.

[37] S. Chong, B. Lee, S. Mitra, R. T. Howe, and H.-S. P. Wong, *Integration of nanoelectromechanical relays with silicon nmos*, *IEEE Transactions on Electron Devices* **59** (2012), no. 1 255–258.

[38] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, *Authentication and authenticated key exchanges*, *Designs, Codes and cryptography* **2** (1992), no. 2 107–125.

[39] C. G. Günther, *An identity-based key-exchange protocol*, in *Advances in Cryptology-Eurocrypt*, pp. 29–37, Springer, 1989.

[40] *Ieee standard specifications for public key cryptography*, .

[41] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[42] T. He, R. Yang, S. Rajgopal, M. A. Tupta, S. Bhunia, M. Mehregany, and P. X.-L. Feng, *Robust silicon carbide (sic) nanoelectromechanical switches with long cycles in ambient and high temperature conditions*, in *Micro Electro Mechanical Systems (MEMS), 2013 IEEE 26th International Conference on*, pp. 516–519, IEEE, 2013.

[43] F. Streller, G. E. Wabiszewski, and R. W. Carpick, *Next-generation nanoelectromechanical switch contact materials: A low-power mechanical alternative to fully electronic field-effect transistors.*, *IEEE Nanotechnology Magazine* **9** (2015), no. 1 18–24.

[44] D. Grogg, C. L. Ayala, U. Drechsler, A. Sebastian, W. W. Koelmans, S. J. Bleiker, M. Fernandez-Bolanos, C. Hagleitner, M. Despont, and U. T. Duerig, *Amorphous carbon active contact layer for reliable nanoelectromechanical switches*, in *2014 IEEE 27th International Conference on Micro Electro Mechanical Systems (MEMS)*, pp. 143–146, IEEE, 2014.

[45] T.-H. Lee, S. Bhunia, and M. Mehregany, *Electromechanical computing at 500 c with silicon carbide*, *Science* **329** (2010), no. 5997 1316–1318.

[46] T. He, F. Zhang, S. Bhunia, and P. X.-L. Feng, *Silicon carbide (sic) nanoelectromechanical antifuse for ultralow-power one-time-programmable (otp) fpga interconnects*, *IEEE Journal of the Electron Devices Society* **3** (2015), no. 4 323–335.

[47] M. Stanisavljević, A. Schmid, and Y. Leblebici, *Reliability of Nanoscale Circuits and Systems: Methodologies and Circuit Architectures*. Springer Science & Business Media, 2010.

[48] M. Tariq Jan, N. Hisham Bin Hamid, M. H. Md Khir, K. Ashraf, and M. Shoaib, *Reliability and fatigue analysis in cantilever-based mems devices operating in harsh environments*, *Journal of Quality and Reliability Engineering* **2014** (2014).

[49] J. I. McCool, *Using the Weibull distribution: reliability, modeling and inference*, vol. 950. John Wiley & Sons, 2012.

[50] U. Schwalke, M. Pölzl, T. Sekinger, and M. Kerber, *Ultra-thick gate oxides: charge generation and its impact on reliability*, *Microelectronics reliability* **41** (2001), no. 7 1007–1010.

[51] A. Arab and Q. Feng, *Reliability research on micro-and nano-electromechanical systems: a review*, *The International Journal of Advanced Manufacturing Technology* **74** (2014), no. 9-12 1679–1690.

[52] M. Berdova, O. M. Ylivaara, V. Rontu, P. T. Törmä, R. L. Puurunen, and S. Franssila, *Fracture properties of atomic layer deposited aluminum oxide free-standing membranes*, *Journal of Vacuum Science & Technology A* **33** (2015), no. 1 01A106.

[53] H. Espinosa, B. Peng, N. Moldovan, T. Friedmann, X. Xiao, D. Mancini, O. Auciello, J. Carlisle, C. Zorman, and M. Merhegany, *Elasticity, strength, and toughness of single crystal silicon carbide, ultrananocrystalline diamond, and hydrogen-free tetrahedral amorphous carbon*, *Applied physics letters* **89** (2006), no. 7 073111.

[54] R. Dewanto, T. Chen, R. Cheung, Z. Hu, B. Gallacher, and J. Hedley, *Reliability prediction of 3c-sic cantilever beams using dynamic raman spectroscopy*, in *Nano/Micro Engineered and Molecular Systems (NEMS), 2012 7th IEEE International Conference on*, pp. 270–273, IEEE, 2012.

[55] A. Barber, I. Kaplan-Ashiri, S. Cohen, R. Tenne, and H. Wagner, *Stochastic strength of nanotubes: an appraisal of available data*, *Composites Science and Technology* **65** (2005), no. 15 2380–2384.

[56] T. S. Slack, F. Sadeghi, and D. Peroulis, *A phenomenological discrete brittle damage-mechanics model for fatigue of mems devices with application to liga ni*, *Journal of Microelectromechanical Systems* **18** (2009), no. 1 119–128.

[57] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, *Measuring real-world accuracies and biases in modeling password guessability*, in *24th USENIX Security Symposium (USENIX Security 15)*, pp. 463–481, 2015.

[58] *ios security guide*, .

[59] *Apple ios hardware assisted screenlock bruteforce*, .

[60] S. Skorobogatov, *The bumpy road towards iphone 5c nand mirroring*, *arXiv preprint arXiv:1609.04327* (2016).

[61] *Apple firmware updates*, .

[62] A. Shamir, *How to share a secret*, *Communications of the ACM* **22** (1979), no. 11 612–613.

[63] A. Juels and M. Sudan, *A fuzzy vault scheme*, *Designs, Codes and Cryptography* **38** (2006), no. 2 237–257.

[64] R. J. McEliece and D. V. Sarwate, *On sharing secrets and reed-solomon codes*, *Communications of the ACM* **24** (1981), no. 9 583–584.

[65] I. Dumer, D. Micciancio, and M. Sudan, *Hardness of approximating the minimum distance of a linear code*, *Information Theory, IEEE Transactions on* **49** (2003), no. 1 22–37.

[66] D. A. Czaplewski, G. A. Patrizi, G. M. Kraus, J. R. Wendt, C. D. Nordquist, S. L. Wolfley, M. S. Baker, and M. P. De Boer, *A nanomechanical switch for integration with cmos logic*, *Journal of Micromechanics and Microengineering* **19** (2009), no. 8 085003.

[67] M. B. Henry and L. Nazhandali, *From transistors to nems: Highly efficient power-gating of cmos circuits*, *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **8** (2012), no. 1 2.

[68] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, *Hybridization of cmos with cnt-based nano-electromechanical switch for low leakage and robust circuit design*, *IEEE Transactions on Circuits and Systems I: Regular Papers* **54** (2007), no. 11 2480–2488.

[69] D. M. Tanner, N. F. Smith, L. W. IRWIN, W. P. Eaton, K. S. HELGESEN, J. J. CLEMENT, W. M. MILLER, S. L. MILLER, M. T. DUGGER, J. A. WALRAVEN, *et. al.*, *Mems reliability: infrastructure, test structures, experiments, and failure modes*, tech. rep., Sandia National Labs., Albuquerque, NM (US); Sandia National Labs., Livermore, CA (US), 2000.

[70] *Targeting system attacks*, .

[71] W. Diffie and M. E. Hellman, *Privacy and authentication: An introduction to cryptography*, *Proceedings of the IEEE* **67** (1979), no. 3 397–427.

[72] D. Kahn, *The codebreakers*. Weidenfeld and Nicolson, 1974.

[73] A. Shamir, *On the generation of cryptographically strong pseudorandom sequences*, *ACM Transactions on Computer Systems (TOCS)* **1** (1983), no. 1 38–44.

[74] *One-time pads (otp)*, .

[75] G. E. Suh and S. Devadas, *Physical unclonable functions for device authentication and secret key generation*, in *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14, ACM, 2007.

[76] Y. Wang, W.-k. Yu, S. Q. Xu, E. Kan, and G. E. Suh, *Hiding information in flash memory*, in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 271–285, IEEE, 2013.

[77] R. P. Brent and H. Kung, *On the area of binary tree layouts*, *Information Processing Letters* **11** (1980), no. 1 46–48.

[78] O. Loh, X. Wei, C. Ke, J. Sullivan, and H. D. Espinosa, *Robust carbon-nanotube-based nano-electromechanical devices: Understanding and eliminating prevalent failure modes using alternative electrode materials*, *small* **7** (2011), no. 1 79–86.

[79] M. Rostami, F. Koushanfar, and R. Karri, *A primer on hardware security: Models, methods, and metrics*, *Proceedings of the IEEE* **102** (2014), no. 8 1283–1295.

[80] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, *A large-scale study of mobile web app security*, *Mobile Security Techologies* (2015).

[81] M. Rostami, J. B. Wendt, M. Potkonjak, and F. Koushanfar, *Quo vadis, puf?: trends and challenges of emerging physical-disorder based security*, in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–6, IEEE, 2014.

[82] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *Fpga intrinsic pufs and their use for ip protection*, in *International workshop on Cryptographic Hardware and Embedded Systems*, pp. 63–80, Springer, 2007.

[83] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, *Dram-based intrinsic physically unclonable functions for system-level security and authentication*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2016).

[84] P. Koeberl, Ü. Kocabaş, and A.-R. Sadeghi, *Memristor pufs: a new generation of memory-based physically unclonable functions*, in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 428–431, EDA Consortium, 2013.

[85] Y. Wang, W. Wen, H. Li, and M. Hu, *A novel true random number generator design leveraging emerging memristor technology*, in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 271–276, ACM, 2015.

[86] J. Rajendran, R. Karri, J. B. Wendt, M. Potkonjak, N. McDonald, G. S. Rose, and B. Wysocki, *Nano meets security: Exploring nanoelectronic devices for security applications*, *Proceedings of the IEEE* **103** (2015), no. 5 829–849.

[87] F. Dabiri and M. Potkonjak, *Hardware aging-based software metering*, in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pp. 460–465, IEEE, 2009.

[88] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu, *Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks*, in *Proceedings of the 21st USENIX conference on Security symposium*, pp. 36–36, USENIX Association, 2012.

[89] C. Yang, B. Liu, H. Li, Y. Chen, W. Wen, M. Barnell, Q. Wu, and J. Rajendran, *Security of neuromorphic computing: thwarting learning attacks using memristor's obsolescence effect*, in *Proceedings of the 35th International Conference on Computer-Aided Design*, p. 97, ACM, 2016.

[90] K. Ng, M. Lee, K. Kwong, and M. Chan, *Diode based gate oxide anti-fuse one time programmable memory array in standard cmos process*, in *Electron Devices and Solid-State Circuits, 2009. EDSSC 2009. IEEE International Conference of*, pp. 457–460, IEEE, 2009.

[91] J. Xiong, Z. Yao, J. Ma, X. Liu, and Q. Li, *A secure document self-destruction scheme: an abe approach*, in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pp. 59–64, IEEE, 2013.

[92] J.-W. Han, M.-L. Seol, Y.-K. Choi, and M. Meyyappan, *Self-destructible fin flip-flop actuated channel transistor*, *IEEE Electron Device Letters* **37** (2016), no. 2 130–133.

[93] N. Banerjee, Y. Xie, M. M. Rahman, H. Kim, and C. Mastrangelo, *From chips to dust: The mems shatter secure chip*, in *Micro Electro Mechanical Systems (MEMS), 2014 IEEE 27th International Conference on*, pp. 1123–1126, IEEE, 2014.

[94] *Self-destructing chips*, .

[95] E. Grochowski and R. E. Fontana Jr, *Future technology challenges for nand flash and hdd products*, *Flash Memory Summit* (2012).

[96] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, *Architecting phase change memory as a scalable dram alternative*, *ACM SIGARCH Computer Architecture News* **37** (2009), no. 3 2–13.

[97] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, *Scalable high performance main memory system using phase-change memory technology*, *ACM SIGARCH Computer Architecture News* **37** (2009), no. 3 24–33.

[98] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, *Overcoming the challenges of crossbar resistive memory architectures*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 476–488, IEEE, 2015.

[99] Z. Deng, A. Feldman, S. A. Kurtz, and F. T. Chong, *Lemonade from lemons: Harnessing device wearout to create limited-use security architectures*, in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 361–374, ACM, 2017.

[100] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, *Use ecp, not ecc, for hard failures in resistive memories*, in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 141–152, ACM, 2010.

[101] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, *A durable and energy efficient main memory using phase change memory technology*, in *ACM SIGARCH computer architecture news*, vol. 37, pp. 14–23, ACM, 2009.

[102] Z. Deng, L. Zhang, D. Franklin, and F. T. Chong, *Herniated hash tables: Exploiting multi-level phase change memory for in-place data expansion*, in *Proceedings of the 2015 International Symposium on Memory Systems*, pp. 247–257, ACM, 2015.

[103] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, *A novel architecture of the 3d stacked mram l2 cache for cmps*, in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 239–249, IEEE, 2009.

[104] R. Bitirgen, E. Ipek, and J. F. Martinez, *Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach*, in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pp. 318–329, IEEE Computer Society, 2008.

[105] Y. Zhou, H. Hoffmann, and D. Wentzlaff, *Cash: Supporting iaas customers with a sub-core configurable architecture*, in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 682–694, IEEE Press, 2016.

[106] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, *Self-optimizing memory controllers: A reinforcement learning approach*, in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pp. 39–50, IEEE, 2008.

[107] L. Zhang, D. Strukov, H. Saadeldeen, D. Fan, M. Zhang, and D. Franklin, *Spongedirectory: Flexible sparse directories utilizing multi-level memristors*, in

Proceedings of the 23rd international conference on Parallel architectures and compilation, pp. 61–74, ACM, 2014.

[108] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens, *Eager writeback-a technique for improving bandwidth utilization*, in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pp. 11–21, ACM, 2000.

[109] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, *Preset: improving performance of phase change memories by exploiting asymmetry in write times*, *ACM SIGARCH Computer Architecture News* **40** (2012), no. 3 380–391.

[110] J. Jeong, S. S. Hahn, S. Lee, and J. Kim, *Lifetime improvement of nand flash-based storage systems using dynamic program and erase scaling.*, in *FAST*, pp. 61–74, 2014.

[111] Q. Li, L. Jiang, Y. Zhang, Y. He, and C. J. Xue, *Compiler directed write-mode selection for high performance low power volatile pcm*, *ACM SIGPLAN Notices* **48** (2013), no. 5 101–110.

[112] M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, *Balancing performance and lifetime of mlc pcm by using a region retention monitor*, in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 385–396, IEEE, 2017.

[113] B. Li, S. Shan, Y. Hu, and X. Li, *Partial-set: write speedup of pcm main memory*, in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–4, IEEE, 2014.

[114] P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi, *Reducing read latency of phase change memory via early read and turbo read*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 309–319, IEEE, 2015.

[115] R. Wang, Y. Zhang, and J. Yang, *Readduo: Constructing reliable mlc phase change memory through fast and robust readout*, in *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pp. 203–214, IEEE, 2016.

[116] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, *Preventing pcm banks from seizing too much power*, in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 186–195, ACM, 2011.

[117] R. Tibshirani, *Regression shrinkage and selection via the lasso*, *Journal of the Royal Statistical Society. Series B (Methodological)* (1996) 267–288.

[118] R. E. Schapire, *The strength of weak learnability*, *Machine learning* **5** (1990), no. 2 197–227.

[119] J. H. Friedman, *Stochastic gradient boosting*, *Computational Statistics & Data Analysis* **38** (2002), no. 4 367–378.

[120] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa, *Ensemble approaches for regression: A survey*, *ACM Computing Surveys (CSUR)* **45** (2012), no. 1 10.

[121] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, *A probabilistic graphical model-based approach for minimizing energy under performance constraints*, in *ACM SIGPLAN Notices*, vol. 50, pp. 267–281, ACM, 2015.

[122] StatTrek.com, "Coefficient of determination." `http://stattrek.com/statistics/dictionary.aspx?definition=coefficient_of_determination`, 2017.

[123] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, *Automatically characterizing large scale program behavior*, in *ACM SIGARCH Computer Architecture News*, vol. 30, pp. 45–57, ACM, 2002.

[124] T. Sherwood, S. Sair, and B. Calder, *Phase tracking and prediction*, in *ACM SIGARCH Computer Architecture News*, vol. 31, pp. 336–349, ACM, 2003.

[125] X. Shen, Y. Zhong, and C. Ding, *Locality phase prediction*, *ACM SIGPLAN Notices* **39** (2004), no. 11 165–176.

[126] E. Duesterwald, C. Cascaval, and S. Dwarkadas, *Characterizing and predicting program behavior and its variability*, in *Parallel Architectures and Compilation Techniques, 2003. PACT 2003. Proceedings. 12th International Conference on*, pp. 220–231, IEEE, 2003.

[127] A. Georges, D. Buytaert, L. Eeckhout, and K. De Bosschere, *Method-level phase behavior in java workloads*, *ACM SIGPLAN Notices* **39** (2004), no. 10 270–287.

[128] A. Jaleel, "Spec cpu 2006 memory workload characterization." `http://www.jaleels.org/ajaleel/`, 2006.

[129] M. Poremba and Y. Xie, *Nvmain: An architectural-level main memory simulator for emerging non-volatile memories*, in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pp. 392–397, 2012.

[130] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, *Overcoming the challenges of crossbar resistive memory architectures*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 476–488, Feb, 2015.

[131] SanDisk, "Sandisk and hp launch partnership to create memory-driven computing solutions." https://www.sandisk.com/about/media-center/press-releases/2015/sandisk-and-hp-launch-partnership,
2015.

[132] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, *Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures*, in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 469–480, IEEE, 2009.

[133] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, *Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory*, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* **31** (2012), no. 7 994–1007.

[134] N. Mishra, J. D. Lafferty, and H. Hoffmann, "Esp: A machine learning approach to estimating application interference." To appear in proceedings of the 14th IEEE-International conference on Automatic Computing, 2017. A preprint available at: `http://people.cs.uchicago.edu/~hankhoffmann/mishra-icac2017.pdf`, 2017.

[135] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, *Adaptive insertion policies for high performance caching*, in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 381–391, ACM, 2007.

[136] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, *Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems*, in *ACM Sigplan Notices*, vol. 45, pp. 335–346, ACM, 2010.

[137] D. Gope and M. H. Lipasti, *Bias-free branch predictor*, in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 521–532, IEEE, 2014.

[138] D. A. Jiménez and C. Lin, *Dynamic branch prediction with perceptrons*, in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pp. 197–206, IEEE, 2001.

[139] E. Teran, Z. Wang, and D. A. Jiménez, *Perceptron learning for reuse prediction*, in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–12, IEEE, 2016.

[140] B. C. Lee and D. M. Brooks, *Accurate and efficient regression modeling for microarchitectural performance and power prediction*, in *ACM SIGOPS Operating Systems Review*, vol. 40, pp. 185–194, ACM, 2006.

[141] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. O'Boyle, *A predictive model for dynamic microarchitectural adaptivity control*, in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 485–496, IEEE Computer Society, 2010.

[142] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, *Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance*, in

Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on, pp. 725–737, IEEE, 2015.

[143] H. Hoffmann, *Jouleguard: energy guarantees for approximate applications*, in *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 198–214, ACM, 2015.

[144] S. R White, N. Sottos, P. Geubelle, J. Moore, M. R Kessler, S. Sriram, E. Brown, and S. Viswanathan, *Autonomic healing of polymer composites*, .

[145] E. T. Thostenson and T.-W. Chou, *Carbon nanotube networks: sensing of distributed strain and damage for life prediction and self healing*, *Advanced Materials* **18** (2006), no. 21 2837–2841.