COMPRESSION ARCHITECTURE FOR BIT-WRITE REDUCTION IN NON-VOLATILE MEMORY TECHNOLOGIES

by

David Dgien

B.S. in Computer Engineering, University of Pittsburgh, 2012

Submitted to the Graduate Faculty of

the Swanson School of Egnineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF EGNINEERING

This thesis was presented

by

David Dgien

It was defended on

March 28, 2014

and approved by

Dr. Kartik Mohanram, Ph.D., Associate Professor

Dr. Steven P. Levitan, Ph.D., Professor

Dr. Jun Yang, Ph.D., Associate Professor

Thesis Advisor: Dr. Kartik Mohanram, Ph.D., Associate Professor

Copyright © by David Dgien

2014

COMPRESSION ARCHITECTURE FOR BIT-WRITE REDUCTION IN NON-VOLATILE MEMORY TECHNOLOGIES

David Dgien, M.S.

University of Pittsburgh, 2014

In this thesis we explore a novel method for improving the performance and lifetime of non-volatile memory technologies. As the development of new DRAM technology reaches physical scaling limits, research into new non-volatile memory technologies has advanced in search of a possible replacement. However, many of these new technologies have inherent problems such as low endurance, long latency, or high dynamic energy. This thesis proposes a simple compression-based technique to improve the performance of write operations in non-volatile memories by reducing the number of bit-writes performed during write accesses. The proposed architecture, which is integrated into the memory controller, relies on a compression engine to reduce the size of each word before it is written to the memory array. It then employs a comparator to determine which bits require write operations. By reducing the number of bit-writes, these elements are capable of reducing the energy consumed, improving throughput, and increasing endurance of non-volatile memories. We examine two different compression methods for compressing each word in our architecture. First, we explore Frequent Value Compression (FVC), which maintains a dictionary of the words used most frequently by the application. We also use a Huffman Coding scheme to perform the compression of these most frequent values. Second, we explore Frequent Pat-

tern Compression (FPC), which compresses each word based on a set of patterns. While this method is not capable of reducing the size of each word as well as FVC, it is capable of compressing a greater number of values. Finally, we implement an intra-word wear leveling method that is able to enhance memory endurance by reducing the peak bit-writes per cell. This method conditionally writes compressed words to separate portions of the non-volatile memory word in order to spread writes throughout each word. Trace-based simulations of the SPEC CPU2006 benchmarks show a 20× reduction in raw bit-writes, which corresponds to a 2-3× improvement over the state-of-the-art methods and a 27% reduction in peak cell bit-writes, improving NVM lifetime.

TABLE OF CONTENTS

1.0 INTRODUCTION 1
1.1 Motivation 2
1.2 Work Overview 3
1.3 Problem Statement 4
1.4 Work Plan 5
1.5 Thesis Organization 6
2.0 BACKGROUND 8
2.1 Non-volatile Memory 8
2.1.1 Phase Change Memory 9
2.1.2 Resistive RAM 9
2.1.3 Spin-Transfer Torque RAM 10
2.1.4 Energy, Endurance, and Latency 11
2.2 Improving NVM Performance 12
2.2.1 Reducing Bit-writes 14
2.3 Memory Compression 15
2.3.1 Non-volatile Memory Compression 18
2.4 Summary 19
3.0 COMPRESSION BASED NON-VOLATILE MEMORY 21

3.1 Reducing Bit-writes 21
3.1.1 Write Method 22
3.1.2 Read Method 23
3.2 Architecture 24
4.0 COMPRESSION METHODS 26
4.1 Frequent Value Compression 26
4.2 Frequent Pattern Compression 29
5.0 WEAR LEVELING 32
6.0 EVALUATION AND RESULTS 36
6.1 Simulation 36
6.2 Freqent Value Compression 37
6.3 Frequent Pattern Compression 39
6.4 Wear Leveling 42
7.0 CONCLUSIONS AND FUTURE WORK 44
7.1 Summary 44
7.2 Contributions 46
7.3 Conclusions 47
7.4 Future Work 48
BIBLIOGRAPHY 49

LIST OF TABLES

1 Frequent pattern compression patterns 30

LIST OF FIGURES

1 NVM memory controller architecture 24

2 Example write operation 25

3 Sample Huffman coding 28

4 Average bit-writes performed using FVC with Huffman Coding. 38

5 Comparison of the number of bit-writes performed. 39

6 Compression ratios of all valid data in memory after trace execution 40

7 Percentage of accesses unable to be compressed 41

8 Comparison of our method with and without wear leveling 42

1.0 INTRODUCTION

The goal of this thesis is to explore the performance of new non-volatile memory technologies and to develop new methods for improving their performance. These new non-volatile technologies, such as phase change memory, resistive RAM, and spin-transfer torque RAM, have been the focus of recent research efforts towards their adoption as a replacement for DRAM in main memory. However, despite their non-volatility and improved density these new technologies are still relatively new in their development and suffer from a number of problems in performance compared to DRAM. Unless these problems are mitigated, as development of DRAM slows main memory will become an increasingly larger bottleneck in the advancement of high performance computing systems. This thesis presents a new write method and memory controller architecture using data compression to reduce bit-writes and improve performance in non-volatile memory technologies.

1.1 MOTIVATION

Over the last few years, computing power has increased substantially, placing greater demands on the technologies that drive these systems. One area with critical issues is main memory, especially in embedded systems where memory performance is critical. DRAM has played a major role in supporting the demands on memory capacity and performance for decades. However, scaling DRAM below 22nm is currently unknown [2], placing limits on its maximum capacity and making it less suitable for next generation main memory. Several new non-volatile memory (NVM) technologies have been considered to address these problems, such as phase change memory (PCM) [16], resistive RAM (ReRAM) [5], and spin-transfer torque RAM (STT-RAM) [15].

However, the performance of these new technologies are affected by a number of problems. Some, such as PCM or ReRAM, suffer from poor cell endurance. In PCM, this is a result of the physical stresses on the phase change material during write operations. The repeated heating and cooling of each cell causes the phase change material to slightly expand and contract. Under enough cycles, these stresses can create gaps in the material or cause the material to break away from the heating element entirely. This ruins the electrical conductivity of the cell, causing it to be "stuck" in one state. PCM and ReRAM cells are only able to sustain on the order of 10^8 writes before failure, compared to the ability of DRAM cells to sustain on the order of 10^{15} writes [33]. Other technologies, such as STT-RAM, suffer from long write latency and high write energy. During write operations, STT-RAM requires a polarized current pulse to switch the spin direction of the free ferromagnetic layer in the magnetic tunneling junction. The length and magnitude of this pulse varies, with faster switching times requiring higher currents, resulting in write latencies 1.25–2$\times$ worse and write energies 5–10$\times$ worse than DRAM [15]. In all of these technologies, limits on the

maximum amount of energy that the device can consume requires the memory to only write a limited amount of data at a time, further impacting the overall write latency of the device. Solutions to these problems must be designed before we can realize widescale adoption of any of these technologies in main memory systems.

1.2 WORK OVERVIEW

In this work, we propose a compression-based architecture to reduce bit-writes and improve write energy, write latency, and write endurance in non-volatile memories (NVM). The proposed architecture, which is integrated into the memory controller, relies on a compression engine and a data comparator which work together to reduce the number of bit-writes that occur during each write access to memory. We discuss two different compression methods to be implemented in the compression engine integrated directly into the memory controller. The first, Frequent Value Compression, relies on a dictionary of the most frequently used values in each application, then compresses these values using a Huffman Coding algorithm [11]. The second method implements the frequent pattern compression (FPC) algorithm [3] to compress the incoming data. The FPC algorithm uses a static pattern table capable of matching a wide range of values and does not require additional memory or application profiling to track frequent values.

When a write access is received by the memory controller, each word is passed through the compression engine to attempt compression of the data. If the word is unable to be compressed, the memory controller writes it "as is" to the memory cells. During the write operation to the NVM cells, the memory controller uses a read-modify-write operation. This operation compares the new (possibly compressed) word against the existing word in memory and updates only the changed bits. Through compression of each word, the

maximum number of bit-writes that are potentially necessary during each write access is reduced significantly. During read accesses, the memory controller checks the status of a tag bit set during write to determine if it has been stored in a compressed format. If it has, the controller again uses the compression engine to decompress the word before forwarding it on the memory bus to the processor. Finally, the proposed architecture utilizes the additional space available through compression to implement an opportunistic wear leveling scheme that conditionally writes the compressed data to the opposite sides of the NVM array. Besides the 10% area increase in the memory controller, the architecture also requires 2 tag bits for every 32-bit word, corresponding to a 6.25% memory overhead.

1.3 PROBLEM STATEMENT

The main problem this thesis addresses, is to design a write method for reducing bit-writes in non-volatile memory technologies. The detailed questions we address are the following.

- How can we use data compression to improve non-volatile memory performance? While compression is frequently used to reduce data size in order to improve storage capacity, the improved density of non-volatile memories compared to DRAM means that this is not a concern. We ask if reducing data size on a smaller granularity can instead reduce bit-writes to improve performance.
- What method of compression is best for this application? While there are many different compression methods and algorithms, we focus on compressing relatively small amounts of data, namely 32 bit words. Because of this not all compression methods are appropriate we need to look at those that are and determine which compression method is ideal for this application.

- Our compression method may have some unintended negative impacts on the performance of the non-volatile memory as a result of concentrating the data within the memory. We need to examine any of the potential problems and develop ways to mitigate them.

1.4 WORK PLAN

To address the problems above, in this thesis we perform the following work:

- We propose a write method for non-volatile memory that reduces bit-flips by attempting to compress each word before we write it to the non-volatile memory array. By compressing the data in the memory on a word by word basis, we can immediately reduce the number of potential bit-writes that may need to be performed during a write access before we even look to see which bits are different and need to be changed.
- We design a modified memory controller architecture to implement our compression based write method. This architecture integrates the necessary compression and decompression engines directly into the memory controller datapath to allow it to seamlessly modify the data as needed. Additional circuitry is added to move the data through the compression and decompression engine, as well as a multiplexer to allow uncompressed read data to bypass the decompression engine entirely.
- We analyze two possible compression methods, Frequent Value Compression (FVC) using Huffman coding and Frequent Pattern Compression (FPC). Frequent Value Compression maintains a dictionary of the most frequently occurring words in the application, which we then encode using Huffman coding to significantly reduce the size of each word. Frequent Pattern Compresson, on the other hand, compresses words based on a set of

patterns. Any data word that matches a pattern is able to be compressed, which allows FPC to compress many more words, albeit not as densely.

- We propose a modification to the write method and architecture to perform intra-word wear leveling. Without this method, compressed words are consistently written to the same portion of the non-volatile memory array, which results in higher write activity to those cells. Our wear leveling method spreads the write activity throughout the word by writing to alternating sides of the memory array. We also compare two different possibilities for deciding when to switch write direction, one based on an internal write access counter, and one based on the number of potential bit-writes performed by each orientation.
- We simulate our methods across a number of benchmarks from the SPEC CPU2006 suite using an in-house simulator. This simulator evaluates the performance of each method by processing memory traces generated from the benchmarks using the Intel Pin instrumentation toolset [18] and counting the number of bit-writes performed during each trace.

1.5 THESIS ORGANIZATION

In Chapter 2, we explore a variety of non-volatile memory technologies that are currently being researched as possible replacements for DRAM as main memory. We then look at many of the problems that are preventing wide-scale adoption of these new technologies. Last we look at the previous work with compression in main memory and non-volatile technologies.

In Chapter 3, we propose our compression based architecture. We describe the write and read methods that allow our method to perform compression and reduce-bit writes. We also describe the modifications made to the memory controller and non-volatile memory array.

In Chapter 4, we introduce the two different compression methods, Frequent Value Compression using Huffman Coding and Frequent Pattern Compression. We describe the backgrounds of each method and describe how each compresses the data words, as well as the differences between the two of them.

In Chapter 5, we describe the reasons for implementing our intra-word wear leveling method, and describe how our method works to spread writes out within each word. We also describe the two options for choosing when to perform wear leveling, the write access counter based method and the potential bit-write based method.

In Chapter 6, we explain our simulation methods and discuss results of our methods. We examine the number of bit-writes performed under Frequent Value Compression with a number of different Huffman coding parameters, and compare this to the performance of the Frequent Pattern Compression method. Additionally we compare the number of peak cell bit-writes performed without our wear leveling method versus the two different wear leveling methods we implement.

Chapter 7 summarizes our work and the contributions of this thesis, draws conclusions, and discusses future work.

2.0 BACKGROUND

In this chapter we discuss the operation of new non-volatile memory (NVM) technologies. Since our method focuses on reducing the number of bit flips, it can potentially be applied to any byte-addressable NVM technology to improve performance. We also provide a brief background on previous efforts to improve the performance of these new NFVM technologies and on the use of compression in SRAM, DRAM and NVM technologies.

2.1 NON-VOLATILE MEMORY

NVMs use a variety of new technologies to store information. Unlike DRAM, which retains information as charge stored in a capacitor, NVM technologies typically store information using different states of a physical system. As a result, the information in these NVMs will remain intact for a far longer time than in DRAM, whereas the capacitors lose information (i.e., charge) in a matter of milliseconds. We discuss the usage and drawbacks of 3 different NVM technologies: phase change memory (PCM), resistive RAM (ReRAM), and spin-transfer torque RAM (STT-RAM),.

2.1.1 Phase Change Memory

PCM technology uses unique properties of chalcogenide glass to store information [16]. This material typically consists of an alloy of germanium, antimony, and tellurium, such as $Gb_2Sb_2Te_5$, called GST. When the material is in a crystalline (SET) state it exhibits low resistance when subjected to an electric current, and when it is in an amorphous (RESET) state it exhibits high resistance. These resistance levels are drastically different, with the resistance in the amorphous state being as much 5 times that of the crystalline state.

To write information to the PCM cell, current is passed through a heating element to melt the chalcogenide material. When performing a RESET operation a short pulse of high current is passed through the heater, raising the temperature of the chalcogenide material above the melting point. This pulse is then quickly terminated to allow the melted material to rapidly cool, resulting in the chalcogenide material being programmed into the amorphous state. In contrast, when performing a SET operation a longer but weaker current pulse is applied to the heating element. This raises the temperature of the chalcogenide material above its crystallization temperature but below its melting point. This allows the chalcogenide material to slowly cool into the crystalline state.

2.1.2 Resistive RAM

ReRAM technology operates on similar principles as PCM, but instead uses the electrical switching properties of metal oxides to store information [5]. ReRAM can be constructed using a variety of metal oxides, such as ZnO or TiO_2. Like PCM, the metal oxide material can be in two different states: a high resistance (RESET) state with the metal oxide material in its regular uniform structure, and a low resistance (SET) state caused by forming oxygen depleted conductive paths through the material.

To perform write operations in ReRAM cells, a voltage difference is applied across the cell to move oxygen ions in the metal oxide into or out of the cell. When performing a SET operation, a positive voltage difference is applied, forcing oxygen ions out of the metal oxide and into the electrode material. These oxygen depleted zones in the metal oxide form conductive metal-only filaments through the material, creating a low resistance state similar to the crystalline state in PCM. During RESET operations, a negative voltage difference is applied, forcing the oxygen ions out of the electrode material and back into the oxygen holes, placing the metal oxide material back into the high resistance state.

2.1.3 Spin-Transfer Torque RAM

STT-RAM technology uses the physical phenomenon of electron spin polarization to store information in a magnetic tunnel junction (MTJ) [15]. An MTJ is created by placing a thin layer of insulating material between two layers of ferromagnetic material. One of these magnetic layers has a fixed magnetic polarization, while the other has a polarization that is free to switch when subjected to a spin-polarized current. The insulating layer prevents the two layers from interfering with each other, but is thin enough to allow electrons to tunnel through so current can flow. In an MTJ, the polarization of the free layer relative to the fixed layer affects the resistance of the cell. When the free layer is polarized parallel to the fixed layer, the cell is in a low resistance (RESET) state, and when the free layer is polarized anti-parallel to the fixed layer, the cell is in a high resistance (SET) state. Note that the resistance states are flipped with their respective write operations compared to PCM and ReRAM.

To perform write operations in STT RAM, a spin polarized current is injected across the MTJ device. By injecting this current from the free layer to the fixed layer, the spin polarization of the current forces the polarization of the free layer to the anti-parallel, or

high resistance, state. Conversely, if this current is injected from the fixed layer to the free later, it forces the polarization of the free layer to the parallel, or low resistance, state.

2.1.4 Energy, Endurance, and Latency

. Read operations in all of these new technologies are comparable to DRAM, consisting of simply placing a small voltage across the device and sensing the current produce to measure the resistance level. However, write operations are the cause of many of their problems and drawbacks.

First, both SET and RESET operations in all of these NVM technologies require a much higher current than DRAM to change the states of the device. Modern memory modules usually include energy consumption limits on their operation to prevent the modules from drawing too much power from the power supply, and to prevent overheating. As a result, devices are restricted to writing a limited amount of data at once, iterating over the memory array to complete writes. This restriction increases the write latency, impacting performance [32]. Second, SET and RESET operations in PCM require long pulses to ensure that the material cools properly, up to hundreds of nanoseconds. As both SET and RESET operations occur simultaneously during a single write, the next write cannot start until all of the longer SET operations are complete, creating a bottleneck in individual write operations [33]. Finally, during the operation of both PCM and ReRAM, frequently changed cells are subject to physical stresses. In PCM, repeated heating and cooling puts thermal stress on the chalcogenide material, while in ReRAM, repeated programming can cause undesirable expansion in the conductive filaments. These stresses can result in poor data retention or can eventually lead to cell failure. This limits the lifetime of these memory cells to around 10^8 write cycles before cell failure [16].

2.2 IMPROVING NVM PERFORMANCE

Across all new NVM technologies, existing solutions use various methods to improve the endurance, latency, and energy problems.

In PCM, for example, one set of solutions tackle the endurance problem through complex data migration or address translation techniques. Start-gap [20] proposes a system for performing wear leveling by moving words within a PCM block. Within each block an additional PCM word is allocated as the "gap" word. After a certain number of write accesses, the gap register is shifted to the next address in the array, by writing the word in that address to the gap address. By combining this moving technique with an address randomization layer, wear to the PCM array can be evened out across the whole array. Security refresh [21] uses address translation techniques to obfuscate the locations of data from potential attackers. A random key is used to swap pairs of addresses across the address space of each PCM block through and xor process. By implementing a two level security refresh system, both within a PCM block and between PCM blocks, the authors are able to both improve memory security as well as perform wear leveling simultaneously. Finally, cache address remapping [25] proposes a hybrid memory system using PCM and a DRAM cache to shield PCM cells from malicious wear-out attacks. CAR exchanges a random set of address bits in the DRAM cache tag with a set of index bits to perform randomized address remapping. This bit exchange, along with the DRAM cache, allows CAR to shield the PCM from malicious attack and reduce wear without requiring remapping table lookups.

Other proposals reduce the write latency directly through architectural improvements. Jiang et. al. propose a Write Truncation and Form Switch method to improve the write performance of multi-level cell (MLC) PCM [13]. Write truncation determines which cells are difficult to write and require many write-and-verify iterations, and which cells do not. This

allows the easy-to-write cells to finish their write operations early. This process is managed using additional ECC cells, which are incorporated into the array using the form switch method to compress each line. Additionally, Yue and Zhu propose a method for preventing write accesses from potentially blocking read accesses in PCM [32]. This method proposes a reorganization of the PCM banks called Parallel Chip PCM (PC2M). This reorganization allows the architecture parallelize each write accesses, as well as divide write acceses up into micro-writes. These parallel micro-writes do not need to be performed contiguously, allowing the much shorter read accesses to a write access by having the remaining micro-writes be performed after the read request is fulfilled.

With STT-RAM, a candidate replacement for SRAM cache or embedded DRAM, architectural improvements similar to those applied to PCM have been considered. Kultursay et al. propose an STT-RAM architecture for main memory that reduces the high energy consumed by write operations [15]. This architecture uses row buffer arrays, similar to [16], to cache data being read from or written to the STT-RAM array. By performing writes just to this row buffer, and tracking whether data in the buffer is dirty or not when it needs to be evicted, this architecture can reduce the number of writes that need to be performed, reducing the overall energy consumed. Zhou et al. also propose an STT-RAM architecture that attempts to reduce the write energy consumed during write operations by implementing early write termination (EWT) [36]. EWT uses special write circuitry to read the value of each cell as it is being written to. If the current value of the cell is the same as the new value to be written, the circuitry cuts off the write current to that cell, preventing it from consuming any more current or energy.

2.2.1 Reducing Bit-writes

Although many of these proposals specifically address the endurance, latency, or energy problems with success, they result in performance penalties in the other categories as a trade-off. This limits the potential improvement these proposals can accomplish. In contrast, there are two recent proposals that improve energy, endurance, and write latency of PCM by simply reducing the maximum number of bits changed during a write access.

The first proposal, Data Comparison Write (DCW) [26] takes advantage of the non-volatility of these new memory technologies by noting that, unlike DRAM, unmodified bits do not require a refresh during write operations. As a result, only those bits that are changing due to overwriting one word in memory with a new one require a bit-write operation and any unchanging bits are simply left alone.

The second proposal, Flip-N-Write (FNW) [7], builds on the DCW method by reducing the maximum number of bits changed during a write access through conditionally "flipping" the data to be written, reducing the number of bit-writes between the old and new data by at least half. However, the improvement in FNW is dependent on both the existing data and the new data to reduce bit-writes. Because our method, like these, focuses mainly on reducing the number of bit-write operations performed during write accesses, we consider these methods to be state-of-the-art and look to further reduce bit-writes beyond FNW.

Simply reducing the number of bit-writes that need to occur per word is very beneficial in that it can potentially improve energy consumption, cell endurance, and write latency, depending on how the memory architecture is configured. Cell endurance improvements from reducing bit-writes are fairly straightforward. By not performing a write operation on cells that are not changing, the cell does not receive any wear and its lifetime is slightly extended. Similarly, write energy consumption improvements are straightforward as well, as only the cells that are undergoing write operations consume energy. How reducing bit writes

improves latency is slightly more complicated. If the memory architecture performs writes using an iterative process where a portion of each word is written at once, such as in [32], or if the memory has a limited power budget such as the one described in [14], then we can see how reducing the bit writes can increase the effective amount of data written at once. For example, if a certain architecture can perform 8 bit-writes in one power cycle and we have 4 words that each only require 2 bit-write operations, then we could potentially write all 4 words in parallel, greatly decreasing the effective time required to write each word through parallelization.

Furthermore, reducing bit-writes is flexible because it does not rely on any particular characteristic of the memory technology other than the requirement that the memory be "bit-addressable". That is, individual bits can be selectively written to, unlike DRAM where all bits in a row are written to in order to perform a refresh. As a result, techniques that just rely on reducing bit-writes for performance gain can be applied to any of the non-volatile memory technologies described in section 2.1.

2.3 MEMORY COMPRESSION

Compression techniques have been investigated widely for classical memory technologies, largely to improve capacity of both main memory systems and last level caches. An analysis of the compressibility of all areas of main memory and caches was able to show that the contents of main memory could be compressed to at least 39% of its original size [19]. IBM developed a commercial main memory compression architecture utilizing a compression method called MXT [24]. MXT utilizes the LZ77 compression algorithm to compress cache lines and dynamically allocates variable size memory sectors to store the data without fragmentation. In [10], the authors propose a block based compression technique for main

memory. In this technique blocks are compressed using a frequent zero-value based compression algorithm, then relocated into a different section of memory. These sections are allocated with different sizes to accommodate different size block, minimizing the amount of unused space while still maintaining the location of each block. In a more recent paper, researchers developed a compression scheme called MemZip [22]. This technique does not improve upon capacity, but instead uses compression to reduce bandwidth to improve energy consumption and latency of the memory. This method is based on the concept of rank subsetting. A cache line is fetched from a single ranks subset and the burst length of the data transfer is a function of the compression factor. A highly compressed cache line requires fewer bursts and therefore saves transfer time and bus energy. However, the methods required to perform the compression and decompression of data usually incur non-negligible latency overheads during the process, and may require complex address translation or space reallocation procedures to account for the variable size of the compressed data within the storage space. For example, an analysis of a number of compression techniques by Yim et. al. showed that those techniques were not sufficient in both sufficiently expanding main memory capacity while alleviating the processor-memory performance gap in traditional technologies [31]. These difficulties in the design of compression systems have impeded the widescale adoption of these techniques in commercial products.

A common technique used for compression in cache and main memory is frequent value compression [28]. Frequent value compression is based on the concept of frequent value locality, examined in depth by Yang [27]. The concept of frequent value locality states that, in memory and caches, a small set of common values occupy a majority of the addresses. Frequent value locality was shown to have applications in a number of areas, including improving capacity in caches [28], reducing the power consumed in buses [29], and improving bandwidth in network-on-chips [37]. For many of these compression applications, such as in

the frequent value cache architecture described in [30], the hardware maintains a dictionary of the words most frequently used in memory, typically storing the results in a local content-addressable memory (CAM). This CAM allows the hardware to quickly determine if a word in a write access is within the set of frequently used words, so it can then replace it with the CAM address of the matching value. This reduces the effective size of the data to $\log_2$(Num. of Values), which can be a significant reduction. The major difficulty with implementing frequent value compression is determining exactly what the frequent values for the given application or working set are. Data profiling can be performed to get this information, either before or during application execution. However, performing profiling before execution only works with a static set of input, while performing profiling during application can incur large overheads if a frequent value needs to be removed from the dictionary.

Another common technique for compression, called frequent pattern compression [3], builds upon the concept of frequent value locality while trying to mitigate some of the issues with it. Frequent pattern compression uses a static set of patterns to determine which values are compressed. The authors of this method chose a set of 7 patterns to compress, based on their analysis of benchmark applications. A more detailed description of the patterns compressed can be found in section 4.2. The main benefit of frequent pattern compression is that it can be implemented entirely using logic gates, without the need for extra memory, CAMs, or application profiling. Additionally, the usage of patterns instead of specific values means that frequent pattern compression can potentially compress many more values compared to frequent value compression. However, for most patterns, frequent pattern compression results in a lower amount of compression per word, and is unable to guarantee compression if none of the values in an application fit one of the patterns. Like frequent value compress, frequent pattern compression has been applied to a variety of areas, including caches [4], main memory [10, 22], and network-on-chips [8].

2.3.1 Non-volatile Memory Compression

Compression for NVMs is a relatively newer area of research. While compression as a mechanism for improving performance in STT-RAM or ReRAM has not been investigated much, as their respective technologies are still under development, compression has been used with limited success within the context of PCM. Because PCM's performance is significantly worse compared to DRAM, compression has a greater potential for closing the processor-memory performance gap to bring PCM's performance closer to DRAM's. The reduced data size as a result of compression can be used to improve performance, energy, and endurance simultaneously as a result of the system reading or writing less data. Additionally, if the remaining free space is not used to store additional data, the overhead of realigning data words within the compressed block can be eliminated. Compression based methods can afford to give up this additional space because most NVMs have scaling comparable to or better than DRAM, maintaining overall improvements in total capacity.

Notably, compression is used in [6] and [17] to create hybrid architecture based on PCM. In [6], the authors design a hybrid architecture that combines DRAM and PCM. The DRAM banks, with their lower latency and dynamic energy requirement, are used to store recently used pages, as a sort of cache. The PCM banks, with their non-volatility and lower static energy, are used to store unused pages, until their data is needed by the processor. A dual-phase compression method is then layered on top, which enhances the DRAM banks by increasing their capacity with the first phase, and enhances the PCM banks by reducing bit-level accesses and performing wear leveling. In [17], the memory system is comprised exclusively of Multi-level cell PCM, and each cache line sized block is compressed before writing. Any lines that can be compressed beyond a 50% ratio are stored in SLC mode cells, which have a lower latency and energy requirement than those cells in 2-bit MLC mode. However, this method requires each entire line to be compressed, meaning each line must be

read and decompressed before data can be extracted from it. Any lines that do not reach the 50% ratio may be written into fewer PCM cells than if they were uncompressed, but still only receive limited performance improvement from utilizing MLC mode cells. Both of these applications implement frequent pattern compression as their compression algorithm. However, they both utilize compression to create additional space then use that space for benefits, rather than taking advantage of compression to directly improve other areas.

Alternatively, frequent value compression is utilized in [23] where the authors use this compression algorithm to directly reduce the number of PCM bit-writes. However, their architecture embeds the compression hardware and dictionary memory directly into the memory chip. While the intent of this method is similar to our own in terms of reducing bit writes, the use of a compression engine embedded in the PCM chip results in unnecessary duplication of compression logic across all chips in the device, increasing overall energy consumption. On the other hand, our proposed method integrates the compression engine entirely in the memory controller and is capable of being applied to general NVM technologies without depending on specific technology features. Furthermore, the FPC algorithm uses a static pattern table capable of matching a wide range of values and does not require application profiling or runtime modifications. Finally, the remaining free space available within each word as a result of compression can be used to further improve endurance using an opportunistic wear leveling scheme, described later in this thesis.

2.4 SUMMARY

In this chapter we introduce background on new non-volatile memory technologies. We describe their operation as well as their drawbacks preventing their commercial adoption as main memory. We look at some of the previous work at mitigating these drawbacks, includ-

ing two general solutions, data comparison write and Flip-n-Write. Both of these methods improve performance by simply reducing the number of bit-writes. Then, we introduce previous work using compression to improve capacity and performance in cache, traditional main memory, and NVM based main memory. We describe two common compression algorithms used in these areas, frequent value compression and frequent pattern compression, and discuss their advantages and disadvantages over each other.

3.0 COMPRESSION BASED NON-VOLATILE MEMORY

In this chapter we introduce our compression based method for reducing bit-writes in non-volatile memories. We describe the write and read methods performed by our method in detail, and propose the modifications to the architecture and memory array necessary for our method.

3.1 REDUCING BIT-WRITES

The main idea of this method is to take advantage of a simple compression scheme to reduce the number of bit-writes that occur as each word in memory is overwritten by a new word. State-of-the-art methods, data comparison write (DCW) [26] and Flip-N-Write (FNW) [7], have been successful in reducing the number of bit-writes during memory operation to improve performance. These methods show how the non-volatility of these new memory technologies can be leveraged by only performing write operations on the bits that need to be changed, while leaving the other bits unmodified. Unlike DRAM, where the entire memory line needs to be written back after each row activation during a read or write operation, refreshing the unchanged bits in NVMs is not necessary, so extra energy or time need not be expended on them. Our method improves upon the state-of-the-art methods by approaching performance improvement from a different angle using data compression. While DCW and

FNW reduce the number of bit-writes through comparison methods, the compression of data words before write allows the proposed method to reduce the number of data bits within each word before performing comparison, resulting in a further reduction in bit-writes.

3.1.1 Write Method

To achieve this reduction in bit-writes, our approach breaks the write access of each memory word into a series of steps. First, an attempt is made to compress the new data to be written using a compression engine integrated directly into the memory controller. This compression engine implements a FPC-based compression method described in Section 4.2. Second, as the compression engine attempts compression of the incoming data, the memory controller reads the existing data currently at the target address within the memory array. Once the compression attempt has completed, the old data and the new data are compared bit by bit to determine the differing bits. If the compression was successful the compressed data is compared against the old data, otherwise the uncompressed data is used. Using the information from this bitwise comparison, only those bits that differ between the old data and the new data are updated and programmed to the NVM cells. If the new data size is smaller than the space it is being written to as a result of compression, the unused bits in the array are ignored during the comparison and update process. Since the reason for comparison is to determine which cells require write operations, the new data is compared to the old data as it exists in memory regardless of its compression state. Finally, the system records the status of the compression process in a compression tag bit in the memory array used to indicate whether or not the data at that address is compressed.

3.1.2 Read Method

During read accesses, these steps are performed in reverse to obtain the original data. First the memory controller reads the word from the NVM cells, including the tag bits. The compression tag bit is checked to determine if the word read is in compressed form or uncompressed form. If the tag bit indicates compression, the word is passed through the decompression engine, and the decompressed result is passed onto the memory bus on its way to the processor. Otherwise, the data bypasses the decompression engine using a multiplexer circuit and the read word is passed directly onto the memory bus, allowing the system to save a few cycles.

3.2 ARCHITECTURE

The proposed architecture is integrated into the memory controller as illustrated in Figure 1. In this architecture the compression engine is integrated directly into the memory controller, allowing data to be seamlessly compressed and decompressed during read and write operations. Extra circuitry is added to facilitate the new data path through the compression/decompression engines, and to allow uncompressed data being read to bypass the decompression engine during read operations.

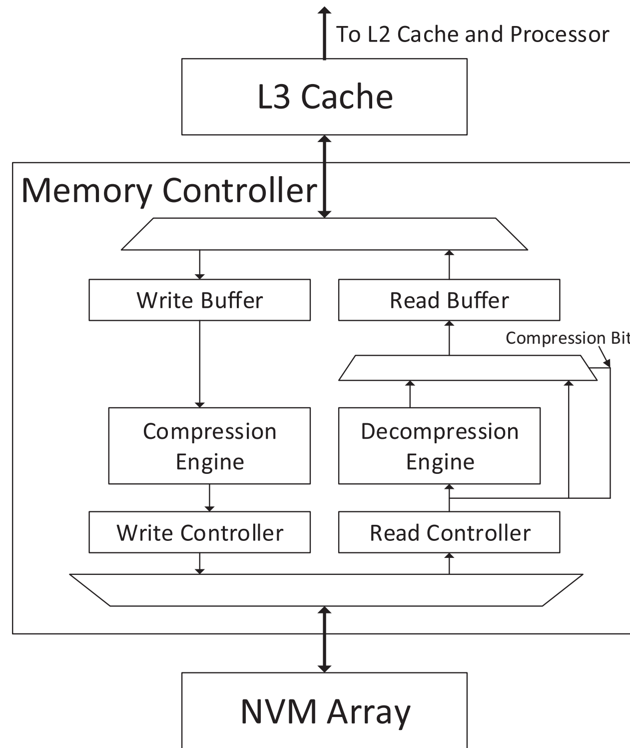


Figure 1: NVM memory controller architecture

The memory array is also modified to provide the tag/status bits to support compression and wear leveling. For each 32-bit word in the memory array, two additional tag bits are added. An example of this, along with an example of a write operation, can be seen in

Figure 2. The first tag bit is used to indicate to the memory controller whether the data from the memory array is compressed or not. This bit is extracted from the word bus in the memory controller and is used as the select signal for the multiplexer that passes the data to the read buffer. The second tag bit is used to support an opportunistic wear leveling technique. This bit is handled within the decompression engine as described in Chapter 5.

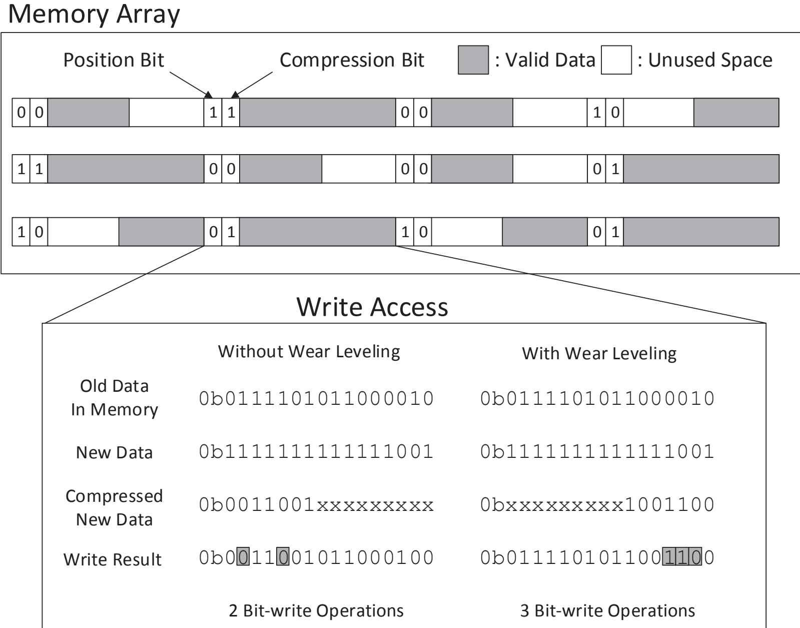


Figure 2: Example write operation

These tag bits together incur a 6.25% capacity overhead overall. This is the same amount of overhead used in the Flip-N-Write method [7], where the authors require 1 tag bit for every 16 bit word. We believe that this is a reasonable amount of overhead for our method.

4.0 COMPRESSION METHODS

The compression algorithm implemented within the memory controller plays an critical part in the performance of the method. In this chapter we discuss two different compression methods for compressing data in order to reduce the number of bit writes: Frequent Value Compression using Huffman Coding, and Frequent Pattern Compression.

4.1 FREQUENT VALUE COMPRESSION

The first compression method that we investigate is Frequent Value Compression (FVC). The main concept behind FVC, frequent-value locality, was first proposed by Zhang et al [34]. This work shows that often during program execution, a small set of distinct values occupy the majority of locations in memory. The authors are able to use this knowledge to create a "Frequent Value Cache" that works along side the main data cache to reduce miss rate. This work has been also been applied to a number of other areas, including a compression based cache [28], low power buses [29], and Network-on-Chip architectures [37].

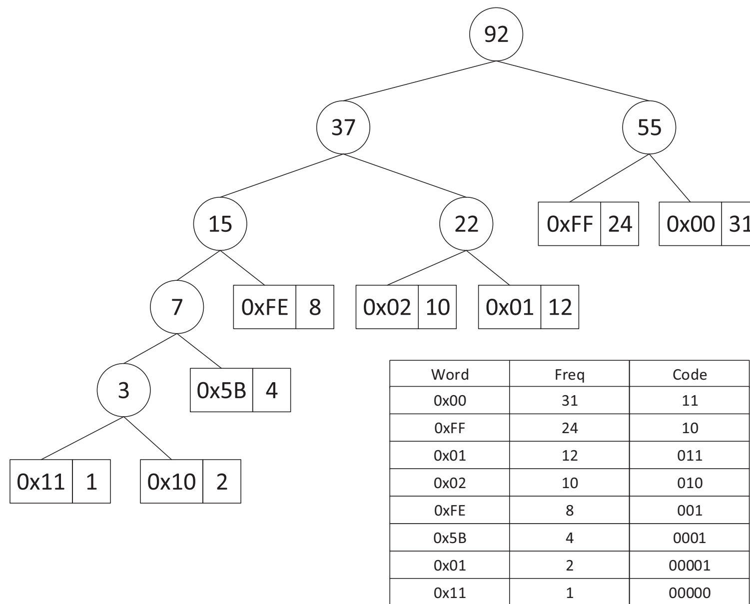
This concept is also used in the NVM domain by Sun et al [23]. The authors of this paper use frequent-value locality to design a compression based main memory architecture for PCM. This frequent value PRAM design uses application profiling, either before or during execution, to determine the set of values that are used most frequently used by that

application. The architecture then stores these frequent values using CAMs to facility easy encoding and decoding during memory operation. As new write accesses are received by the memory module, they are checked against the frequent value CAM to determine if they match a frequent value. If they do, instead of writing the full value, the module writes the address of the frequent value in the CAM instead. Because the number of bits of the is only $\log_2$(Num. of Frequent Values), the number of bit-writes can be greatly reduced. Our method is similar to that used in [23], however we store the frequent values and perform compression within the memory controller, instead of within the memory modules themselves.

Additionally, we believe that there is some room for improvement in the compression method used by a frequent value based architecture. Instead of using the simple address based method for data compression, we chose to use a Huffman coding based method. Huffman coding was created in the 1950s by David Huffman as an efficient binary coding scheme [11]. It is a frequency based, variable length coding generated by constructing a binary tree based on the frequency of each value. In this tree, leaf nodes with more frequent values are shallower than those with less frequent values, which results in code word sizes that vary based on the value frequency. As a result, the average number of bits in the encoded data set is minimized, which is beneficial for our goal of reducing the number of bit-write operations.

To construct a Huffman coding, first a sorted list of the most frequent values used during application execution, along with the frequency with which each value appears, is generated. From this table, a binary tree is constructed from the bottom up, starting with the least frequent values. The two least frequent values are removed from the list, and are used to create a parent node, with the two values as its children nodes. This parent node is given its own frequency value equal to the sum of the frequencies of its two children, then is added back to the list of values. The value list's order is maintained with the new parent node being

placed in the list according its own frequency. Once the list of values and nodes is emptied, we can use the resulting binary tree to construct the codes that represent each of the values. By performing a pre-order traversal of the tree and having left branches append a '0' to the code and right branches append a '1', we can progressively build up a prefix-free code word for each of the values in our frequent value list. An example Huffman tree constructed using 1-byte words can be seen in Figure 3.



Word	Freq	Code
0x00	31	11
0xFF	24	10
0x01	12	011
0x02	10	010
0xFE	8	001
0x5B	4	0001
0x01	2	00001
0x11	1	00000

Figure 3: Sample Huffman coding

Once the frequent values and their respective codewords have been determined for the application, they need to be incorporated into the compression engine integrated into the memory architecture. To achieve this, we use a set of CAMs to store the frequent values and their respective code words, similar to the method used in [23]. To facilitate this, we construct the Huffman coding tree using just the N most frequent values, where N is the size of the CAM. Compression is performed simply by matching the incoming write data against the saved frequent values using a simple CAM, and replacing matching values with

their codewords. Decompression is similarly simple, with the same process occurring using a ternary CAM with the unused bits in the code words set to don't-care values. A ternary CAM is perfect for this application, as the Huffman codes are prefix-free, meaning that no codeword is a prefix of any other code word.

Frequent value compression is beneficial as a result of its simple implementation in hardware, and fast compression and decompression speeds. The authors of [23] report that circuit level simulations for a CAM lookup operation show an access latency of just one cycle on a CAM size of up to 128 values. Additionally, using Huffman coding as our encoding technique should result in a further reduction of bit-writes, at the expense of requiring profiling to be performed on the application before execution.

4.2 FREQUENT PATTERN COMPRESSION

A second option for data compression in memories is Frequent Pattern Compression (FPC) [3]. FPC is a significance-based compression technique that matches the incoming data against pattern classes to achieve compression. FPC was originally proposed for use in the L2 cache to expand its effective capacity, thereby improving performance and reducing miss rate.

In FPC, 32-bit words are compressed based on a set of patterns, rather than specific values. Table 1 describes the patterns used by FPC.

These patterns were selected based on their high frequency of occurrence in many integer and commercial benchmarks [3]. In contrast to FVC, FPC requires no application profiling. Instead, the compression and decompression engines for FPC are implemented entirely using CMOS logic. When a word is input into the compression engine, the content of the word is checked to determine if it matches any of the frequent patterns. If a match is found, the engine encodes the word appropriately and prepends the matching prefix. This 3-bit prefix

Table 1: Frequent pattern compression patterns

Prefix	Pattern Encoded	Example	Compressed Example	Encoded Size	Value Space
000	"Zero Run"	0x00000000	000	0 bits	1
001	4-bit Sign Extended	0x00000007	0010111	4 bits	15
010	1-byte Sign Extended	0xFFFFFFB6	01010110110	8 bits	240
011	Halfword Sign Extended	0x00005432	0110101010000110010	16 bits	65280
100	Halfword, padded with a zero Halfword	0x54320000	1000101010000110010	16 bits	65535
101	Two Halfwords, each a byte sign extended	0xFFB60036	1011011011000110110	16 bits	65025
110	Word consisting of four repeated bytes	0x20202020	11000100000	8 bits	254

is used during decompression to classify which pattern the compressed value matches, so the decompression engine can decode the value.

The main benefits of FPC come from the wide range of values that it can compress, and the simple way that these values can be described. For example, most integers used in programs can be expressed in just 4, 8, or 16 bits, despite being stored using 32 bits. Table 1 shows that the total number of values that can be compressed with FPC is almost 200,000, which is a much wider range of values than most dictionary based compression schemes, e.g., frequent value compression [23]. Additionally, the pattern table used in FPC is static and does not need to be generated through application profiling or at runtime. This means that the compression engine can be realized with relatively simple logic, embedded directly into the memory controller, and requires no internal memory to store a value dictionary. This simplifies the implementation of the compression engine and keeps the overhead associated with the compression engine low.

The FPC compression engine takes up only 10% additional area in a typical memory controller, requires 3 additional cycles to perform compression, and 5 additional cycles to perform decompression [17]. The reduction in bit-writes generated by the proposed architecture ensures that more words can be written in one write operation within the power budget limit [33], thereby compensating for the 3 extra cycles of delay incurred by the FPC compression engine. Even though the FPC compression engine causes 5 extra cycles of delay in each read operation, the widely used "open-page" policy in the row buffer of the memory architecture significantly reduces the number of read operations that exactly reach the cell array. Thus, the timing overhead in the read operation can be reduced by increasing the size of row buffer [12].

Like FVC, FPC has a simple implementation that can be realized entirely in hardware. While FVC's value list is much more tailored to the specific application and may have better word compression ratios, FPC is capable of compressing a much wider range of values which can result in more words being compressed. These minimal overheads and simple implementation combined with good compression over a wide range of frequent values also make FPC an ideal compression method for the proposed architecture.

5.0 WEAR LEVELING

Although the proposed method is capable of improving write latency and energy through reducing the number of bit-writes, endurance benefits can be limited. While the comparison step of our method does mitigate cell endurance problems in general, the compression method creates some restrictions that can impact endurance. In both methods used to compress the data, the location of the first bit and the orientation of the data is important for determining the length of the relevant data, as different patterns are reduced to different lengths in FPC, and different frequent values are compressed to different lengths in FVC. In order to ensure proper decompression, the decompression engine must know exactly where the starting bit is in the bit array to determine the compressed data length. To preserve this, compressed data is always aligned to the most significant portion of the NVM word. The result of this is that the most significant cells of each NVM word are affected by write operations during every write access, while the least significant cells are only affected by writes of uncompressed data and are relatively underutilized. We believe there is room for improvement to shift some of the writes from the most significant to the least significant portion of the NVM word.

Fine-grain and coarse-grain wear leveling for PCM was introduced in [35]. In fine-grain wear leveling, the bit-writes are distributed evenly among the row by using a shift mechanism. In coarse-grain wear leveling, memory pages are grouped into segments. Segments with more bit-writes are swapped with segments with fewer bit-writes. However, both approaches require additional information such as a remapping table, resulting in significant memory

space overhead. Another wear leveling method, start-gap, was proposed [20]. Instead of using a mapping table, start-gap utilizes two pointers to determine how to swap pages, resulting in less space overhead. However, this approach cannot distribute the writes evenly in some memory regions when the memory access has uneven write patterns. A wear rate leveling approach was presented [9], which considers the endurance variation of PCM cells and puts high priority on protecting the "weak" PCM cells. Similar to the methods in [35], memory space overhead in [9] is also significant due to the implementation of the remapping table. These state-of-the-art solutions focus on implementing wear leveling across addresses in NVM, while our method is unique in that it performs wear leveling within each word, using the free space available from compression.

To accomplish opportunistic wear leveling in our architecture, a modification to the compression process is proposed. This modification writes compressed data to both sides of the NVM word at different times, allowing the wear on the NVM cells to be evened out across the whole word. During write accesses, instead of always writing the data to the most significant portion of the NVM word, the data is conditionally flipped horizontally so that the most significant bit is now the least significant bit, and is written to the least significant portion (lower half) of the NVM word. Flipping the data ensures that the memory controller always knows the location of the prefix, so it can determine the compressed data length during decompression. This method works well for both compression methods, because they both are capable of reducing the data size significantly. For example, all of the FPC patterns reduce the data size to at least 16 bits (19 with the prefix), which is about half the size of 32-bit NVM word. An example of a write operation with this wear leveling process is illustrated in Figure 2.

To manage this wear leveling process, some modifications to the memory controller and NVM array are necessary. First, a second "position" tag bit is added to each word. This tag

bit indicates to the memory controller the side of the NVM array that the data is currently stored on. Second, additional circuitry is added to the memory controller to determine whether the compressed data should be written normally or flipped. Once the memory controller has determined this information, it sets the position tag bit appropriately, with a "0" indicating normal orientation and a "1" indicating flipped orientation. If the data to be written is not compressed, then the position bit is unmodified, as data position is irrelevant to uncompressed data.

During read accesses, after the memory controller detects that the data is compressed, it reads the position bit to determine the location of the data in the array. If the position bit indicates that the data was written to the lower half of the NVM word, then the data read by the memory controller is flipped to restore the prefix to the most significant bits, and the data is passed to the decompression engine to be decompressed. If the data read is not compressed, then memory controller just ignores the position bit.

Two different methods for determining which orientation to write the data in are tested. The first method uses an internal write-access counter to determine which orientation to write the data. This counter increments every time a write access is received by the memory controller, and uses an indicator to tell the memory controller which orientation to write the data. Once a certain threshold of write accesses has been received, the indicator switches between "flipped" and "normal" (or vice versa) and the counter resets. Before the comparison and write step of the proposed write method, if the data to be written is compressed, the memory controller checks the position indicator to determine which side of the memory array data to write to. If the counter indicates "flipped", then the (horizontally flipped) data should be written from the least significant cell toward the most significant cell, on the lower half of the NVM word. Conversely, if the counter indicates "normal", then the data should be written normally, from the most significant cell toward the least significant cell,

on the upper half of the NVM word. Finally, if the counter indicator has changed since the last write operation to this address, the position tag bit is updated to match the counter indicator.

The second method tested uses the number of potential bit-writes to determine orientation, implementing a more opportunistic wear leveling method During the comparison step of the proposed write method, the additional circuitry in the memory controller compares the new data in both the normal (most significant) orientation, and the flipped (least significant) orientation, against the existing data. The memory controller uses these comparisons to calculate the number of bit-flips that would be required to perform both write operations. The circuitry then compares these results to determine which orientation would result in a fewer number of bit-flips. The orientation with the fewer number of bit-flips is then selected and written to the location in memory, with the position tag bit set appropriately.

Although the differences between these two methods are minor, there are some trade offs between them. The main problem for the bit-write based method is that we cannot guarantee that any wear leveling will actually occur. In the worst case scenario, the optimal orientation may always be the same, resulting in no improvement in wear leveling performance. However, while the counter based method guarantees that wear leveling will occur, it may result in an increase in overall bit-flips, which is not ideal.

6.0 EVALUATION AND RESULTS

6.1 SIMULATION

The proposed architecture is evaluated using an in-house trace-driven simulator. The traces used are generated from benchmarks in the SPEC CPU2006 [1] benchmark suite. These benchmarks reflect a variety of real integer and floating-point based workloads used by modern computing systems. The memory traces from the benchmarks are recorded using the Intel PIN Binary Instrumentation Tool [18] on a machine running a 2.3GHz Intel Core i7 CPU. Our tool captures memory accesses from the processor and simulates a typical cache system, recording only those accesses sent to main memory. This tool simulates a separate 32 KB, 4-way associative I-cache and 32 KB 8-way associative D-cache at L1, with a shared 256 KB 8-way associative L2 cache and an 8 MB 16-way associative L3 cache. When gathering the traces, the benchmarks are first run through 1 billion instructions to avoid memory accesses from program initialization; they are then run until 1 million memory write operations have been recorded.

We run simulations to evaluate the performance characteristics of frequent value compression under different parameters, and compare the performance improvement of our compression based methods with two state-of-the-art methods, Flip-N-Write (FNW) [7] and data comparison write (DCW) [26]. As explained in Chapter 4, these two methods are similar to ours because they also perform a bitwise comparison to only update changed bits. For all

methods, our simulator compares the new data against the existing data in the data structure before replacing it to calculate and record which bits are different and would undergo a bit-write operation in a real device.

6.2 FREQENT VALUE COMPRESSION

Our simulator evaluates FVC using Huffman coding with a two step process. First, the simulator performs application profiling by taking a preliminary pass of the memory trace, performs the raw writes a data structure representing the contents of memory, and records the number of times each data word appears in memory during trace execution. Once a frequency analysis of the memory trace is complete, the simulator constructs the Huffman tree using the Huffman coding algorithm described in Section 4.1, and uses this to determine the Huffman codes for each of the words in the tree. The data words and their respective code word are saved in a data structure to facilitate easy encoding and decoding during the rest of the simulation. In the second step, the simulator erases the contents of the memory data structure and begins a second pass of the trace file. In this pass, the simulator processes each write access, this time performing compression on each word if possible by exchanging it with its Huffman codewords. The modified data is then stored in the memory data structure, and any bit changes are recorded to determine the performance of this method.

We evaluated FVC with a number of different Huffman tree sizes from 16 to 256 of the most frequent values. Additionally, we also ran evaluations where the tree generated by the simulator is forced to be balanced. By placing the most frequently occurring values in a balanced tree while still generating codewords using the tree-traversal method used in the Huffman coding algorithm, the resulting codeword values have a constant length. This configuration mirrors the address based frequent-value PRAM architecture described in [23]. The results of this analysis can be seen in Figure 4.

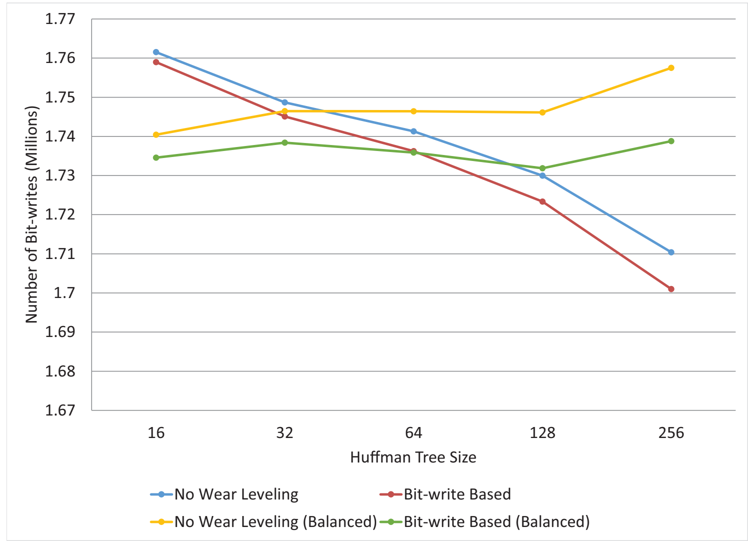


Figure 4: Average bit-writes performed using FVC with Huffman Coding.

We can see a distinct trend in average number of bit-writes among the un-balanced runs, with larger trees allowing for fewer bit-writes. With the balanced tree however, the tree size seems to have little effect on the overall number of bit-writes. We believe that this is because a greater number of accesses contain a fewer number of values, and balancing the tree to simulate the frequent-value method negates the benefits of letting some codewords have shorter lengths (and thus fewer bit-writes) than others. Additionally, we note that for both the Huffman coding simulations and balanced tree simulations, the introduction of our opportunistic wear leveling scheme resulted in an overall lower number of bit-writes compared to no wear leveling. This is expected, as our opportunistic wear leveling scheme ensures that the orientation with fewer bit-writes is always chosen, whereas without wear leveling the system does not have this choice.

6.3 FREQUENT PATTERN COMPRESSION

We evaluate FPC using a similar procedure to the FVC evaluation. However, because FPC does not require application profiling, the simulator does not perform this step of the simulation. Instead, the simulator is able to begin on the second step of the process, and requires only a single pass of the trace file to be able to simulate the compression method and calculate the number of bit changes occurring during the course of the trace. Additionally, with the compression patterns hardcoded into the FPC technique, there are no parameters to adjust as in the FVC technique. Instead, we compare the results of FPC directly to those of DCW, FNW, and FVC.

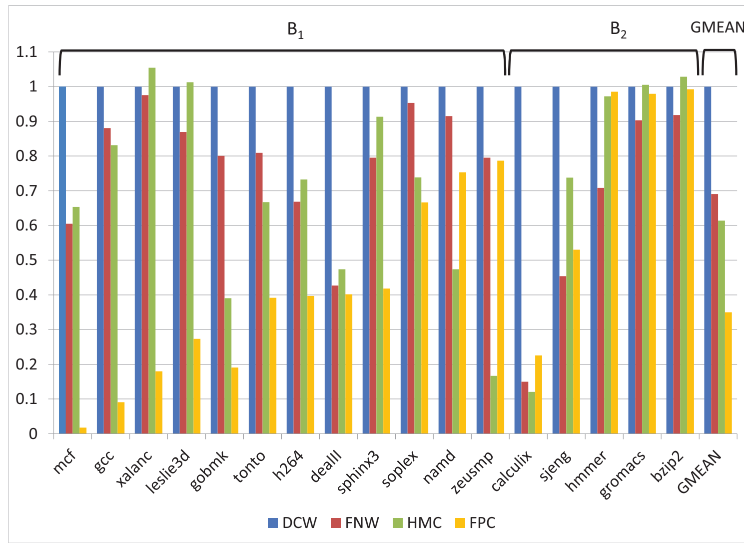


Figure 5: Comparison of the number of bit-writes performed.

Our simulation reports the number of bit-writes performed within each trace, the results of which can be seen in Fig 5. The number of flips is normalized to the DCW method, and we used the FVC results corresponding to the 128 value Huffman tree. FPC shows the greatest reduction in bit writes, with FVC showing a small amount of improvement over

FNW overall. For FPC, improvement is shown in number of bit-write operations performed in the majority of benchmarks (set B_1), especially in the gcc (89% reduction in bit-writes) and mcf (95% reduction in bit-writes) benchmarks. On average, FPC reduces the number of bit-write operations by 3× over DCW and 2× over FNW across all benchmarks. For FVC, on the other hand, number of bit-writes are generally much closer to the number performed by FNW with FVC reducing the number of bit-write operations by 39% over DCW and just 9% over FNW in the average case.

Under the FPC algorithm a few of the benchmarks do report an increase in the number of bit-write operations (set B_2): bzip2, calculix, gromacs, hmmer, and sjeng. We can obtain some insight as to why these benchmarks report an increase in bit-write operations in Figure 6 and Figure 7.

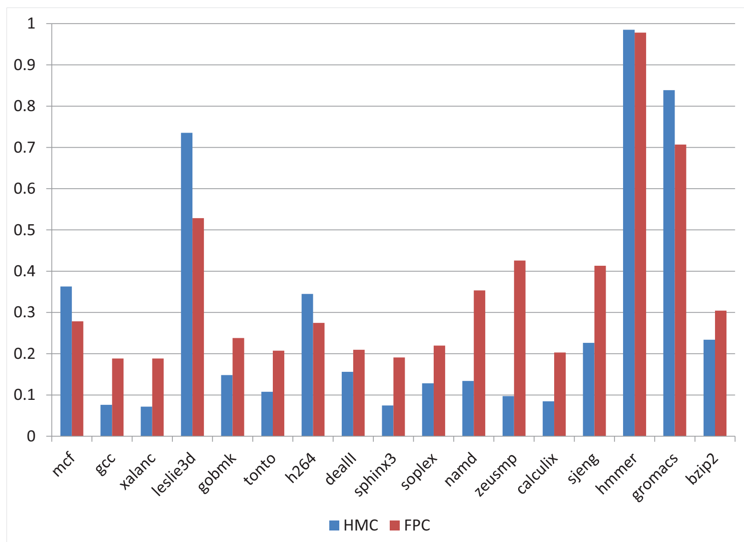


Figure 6: Compression ratios of all valid data in memory after trace execution

These figures report the compressibility of each benchmark under the FPC algorithm both in terms of number of uncompressed write accesses performed, and the final compression ratio of the data in memory at the end of simulation for each benchmark. High values in

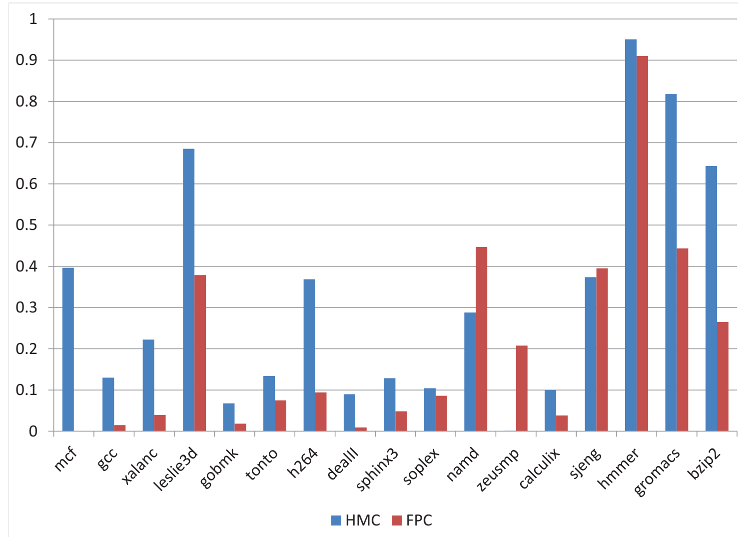


Figure 7: Percentage of accesses unable to be compressed

either of these areas indicate a poor fit to the respective algorithm. Here, most benchmarks reporting an increase in bit-writes also report a high percentage of uncompressed accesses or a low compressed size. Hmmer and gromacs in particular are not very compressible under FPC, with hmmer only compressing 9% of accesses and gromacs compressing the data in memory by only 30%, resulting in poorer performance. Additionally we gain some insight as to why FVC produces poorer results compared to FPC from these graphs. FVC typically has a much larger percentage of accesses that are unable to be compressed than FPC across many of the benchmarks, which contributes to its poorer performance. Despite the problems with compression in the five benchmarks in set B_2, their performance decrease for FPC over FNW remains small.

6.4 WEAR LEVELING

The impact of the wear leveling method on bit-writes to specific cells in each memory word is also evaluated. The simulator reports the number of bit-write operations that occur to each bit index, over the course of the simulation and across all words. This data is used to determine the number of bit-flips occurring at the bit index with the maximum number of writes for each benchmark. The cells that experience the most bit-flips will typically be the first to fail, so the wear leveling technique attempts to reduce this number, extending the lifetime of the whole word. The results of this analysis can be seen in Figure 8.

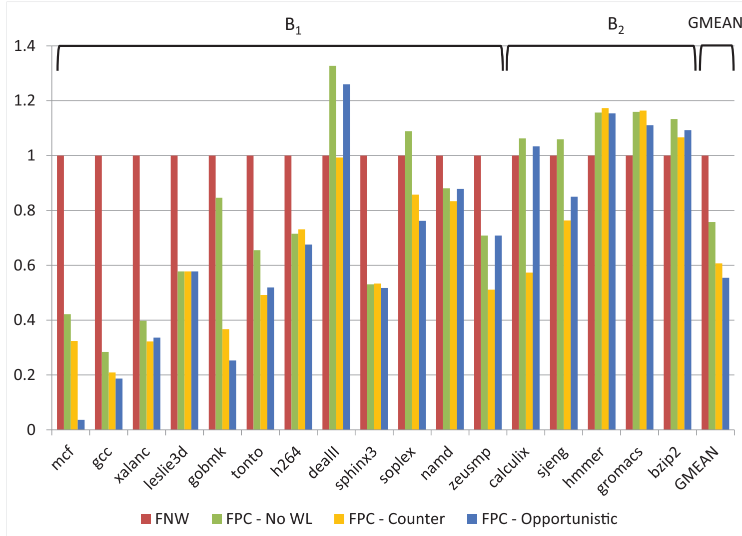


Figure 8: Comparison of our method with and without wear leveling

We compare improvements when implementing our opportunistic wear leveling method on top of FPC, and normalize the results to the case with no wear leveling implemented. Generally, there is a reduction in the maximum number of bit-flips across the benchmarks when implementing the wear leveling technique. Again, we see similar patterns as with the bit-flip results, where those benchmarks that exhibit poor compressibility (leslie3d, hmmer,

gromacs, bzip2) also respond poorly to this wear leveling technique. However, the performance decrease is very minimal (at most 3%) compared to the endurance gains in the average case, with the peak cell bit-writes being reduced by 27%. As the failure of one cell results in failure for the whole word, our wear leveling method is capable of extending lifetime by reducing maximum number of bit-writes to specific cells.

7.0 CONCLUSIONS AND FUTURE WORK

We have presented a novel compression based write method and architecture to reduce the number of bit-writes performed during write operations in new non-volatile memory technologies. We explore the use of two different compression methods for performing compression of each word, Frequent Value Compression using Huffman Coding and Frequent Value Compression. Finally we add an intra-word wear leveling method to spread bit-writes out within each non-volatile memory word and reduce peak cell bit-writes.

In the rest of this chapter, we summarize the work in this thesis again. Then we list the main contributions of our work and the important conclusions we can draw from experiments. Finally, we introduce some remaining problems in this area and a few of our ideas to extend our work.

7.1 SUMMARY

First, in Chapter 2 we introduce background information on 3 new non-volatile memory technologies: phase change memory, resistive RAM, and spin-transfer torque RAM. We describe how each of these work and what makes them ideal as possible replacement for DRAM as main memory as well as their potential drawbacks. We describe past work for improving the performance of non-volatile memory technologies, including two techniques

for generally reducing the number of bit-writes performed, data comparison write and Flip-n-Write. Finally we introduce past work for performing compression in both traditional main memory and in non-volatile memories and describe two general compression algorithms used, frequent value compression and frequent pattern compression.

Chapter 3 introduces our compression based architecture and write method for reducing bit-writes in non-volatile memories. We describe the details of the write and read methods used to compress each word in memory accesses as they are received by the memory controller, and how these methods are capable of reducing bit-flips. We introduce the modifications to the memory controller, including the integration of the compression and decompression engines, as well as the addition of the two tag bits to each word in the memory array.

In Chapter 4 we discuss the two different compression methods we explored for compressing data in our method. In this work we chose to compare Frequent Value Compression with Huffman Coding versus Frequent Pattern Compression. We explain how each works and compare the benefits and potential downsides of each.

In Chapter 5 we propose a wear leveling method to spread bit-writes out within each word and further improve the non-volatile memory endurance. We discuss two different options for choosing when to perform the wear leveling switch, one based on a write access counter and one based on the number of potential bit-writes performed.

Chapter 6 presents our simulation methods and results. We discuss the in-house memory simulator we developed and how it counts bit-flips to compare performance between the various compression methods. Overall we find that frequent pattern compression produces the greatest improvement, reducing overall bit-writes by 3× over Data Comparison Write and 2× over Flip-n-Write. Additionally, our wear leveling method is able to reduce peak cell bit-writes by 45% over Flip-n-Write and by 27% over our own method without wear leveling.

7.2 CONTRIBUTIONS

The contribution that our work makes in the area of improving performance in new non-volatile memory technology for use in main memory can be summarized as the following:

- We determine that reducing the number of bit-writes performed during write accesses in non-volatile memories can potentially improve cell endurance, energy consumption, and write latency simultaneously.
- A write method utilizing data compression to reduce bit-writes is developed. The modified read and write methods are described in detail. We also describe the modified memory controller architecture used to implement our methods.
- We examine two different compression methods as possibilities for performing compression in our method, Frequent Value Compression with Huffman Coding and Frequent Pattern Compression. The possible benefits and drawbacks of each method are explored.
- An intra-word wear leveling method is developed to further improve non-volatile memory endurance. Compressed data is written to alternating sides of the non-volatile memory array in order to spread bit-writes out within each word and reduce peak cell bit-writes.

7.3 CONCLUSIONS

Based on the exploration of this work, a set of important conclusions can be drawn:

- Our compression based write method and memory architecture is successful at reducing bit-writes by compressing data within each word. Performing fewer bit-writes during write accesses results in less energy consumed and an improvement in endurance in the non-volatile memory, as well as allowing for more words to potentially be written at one time improving throughput.

- Changing the size of the dictionary impacts frequent value compression performance, with a larger dictionary allowing for better compressing and fewer bit-writes. Additionally, using Huffman coding allows for better performance over a balanced coding method.

- Frequent pattern compression results in significantly fewer bit-writes compared to frequent value compression. An analysis of the benchmarks under two compression methods shows how frequent pattern compression is able to compress more accesses and results in a better compression ratio compared fo frequent value compression to achieve this reduction in bit-writes. Additionally, because frequent pattern compression is easier to implement and its higher overheads can be mitigated, we believe that it is an ideal compression algorithm for our method.

- Our wear leveling method is able to successfully reduce peak cell bit-writes by writing compressed data to alternating sides of the non-volatile memory array. While both the counter based method and bit-write based method are both able to reduce peak cell bit-writes successfully, the bit-write based method results in slightly better results, and is also capable of reducing overall bit-writes, leading to better performance.

7.4 FUTURE WORK

The work presented in this thesis makes some major contributions, but we have some room to take it further. Future work can be summarized by the following points:

- Because of the limited endurance of many non-volatile technologies, memory modules often include additional overhead for error correcting codes (ECC). Because a minor change in the data can have a major impact on the contents of the ECC, our compression method may have a larger effect on these ECC cells. An investigation of potential impacts on ECC by our method, or ways or method can be adapted to ECC should be taken.
- Many new non-volatile memories also use multi-level cells (MLC) to further improve density. However, our method assumes the use of single-level cells (SLC) as each bit-write reduction equates to one cell that no longer needs to perform a write operation. Using MLC memory with our method means that a cell has a much higher probability of being changed, which means that many of the benefits could be negated. The impacts of using our method with MLC memory should also be examined.

BIBLIOGRAPHY

[1] SPEC CPU2006, 2006.

[2] International technology roadmap for semiconductors, 2011.

[3] A. Alameldeen and D. Wood. Frequent pattern compression: A significance-based compression scheme for L2 caches. Technical report, Univerity of Wisconsin-Madison, 2004.

[4] A.R. Alameldeen and D.A. Wood. Adaptive cache compression for high-performance processors. In *Proc. Intl. Symposium on Computer Architecture*, 2004.

[5] I.G. Baek et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses. In *Proc. Intl. Electron Devices Meeting*, 2004.

[6] Seungcheol Baek et al. A dual-phase compression mechanism for hybrid DRAM/PCM main memory architectures. In *Proc. Great Lakes Symposium on VLSI*, 2012.

[7] S. Cho and H. Lee. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proc. Intl. Symposium on Microarchitecture*, 2009.

[8] R. Das et al. Performance and power optimization through data compression in network-on-chip architectures. In *Proc. Intl. Symposium on High Performance Computer Architecture*, 2008.

[9] J. Dong et al. Wear rate leveling: Lifetime enhancement of PRAM with endurance variation. In *Proc. Design Automation Conference*, 2011.

[10] M. Ekman and P. Stenstrom. A robust main-memory compression scheme. In *Proc. Intl. Symposium on Computer Architecture*, 2005.

[11] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. IRE*, 1952.

[12] B. Jacob et al. *Memory systems: Cache, DRAM, and disk*. Morgan Kaufmann Press, 2007.

[13] L. Jiang et al. Improving write operations in MLC phase change memory. In *Proc. Intl. Symposium on High Performance Computer Architecture*, 2012.

[14] Lei Jiang et al. Fpb: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory. In *Proc. Intl Symposium on Microarchitecture*, 2012.

[15] E. Kultursay et al. Evaluating STT-RAM as an energy-efficient main memory alternative. In *Proc. Intl. Symposium on Performance Analysis of Systems and Software*, 2013.

[16] B. Lee et al. Architecting phase change memory as a scalable DRAM alternative. In *Proc. Intl. Symposium on Computer Architecture*, 2009.

[17] H.-G. Lee et al. A compression-based hybrid MLC/SLC management technique for phase-change memory systems. In *Proc. IEEE Annual Symposium on VLSI*, 2012.

[18] C.-K. Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. In *Proc. Conference on Programming language design and implementation*, 2005.

[19] N. Mahapatra et al. A limit study on the potential of compression for improving memory system performance, power consumption, and cost. *Journal of Instruction-Level Parallelism*, 2005.

[20] M. Qureshi et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proc. Intl. Symposium on Microarchitecture*, 2009.

[21] N. Seong et al. Security refresh: Protecting phase-change memory against malicious wear out. *IEEE Micro*, 2011.

[22] Ali Shafee et al. MemZip: Exploring unconventional benefits from memory compression. In *Proc. Intl. Symposium on High Performance Computer Architecture*, 2014.

[23] G. Sun et al. A frequent-value based PRAM memory architecture. In *Proc. Asia and South Pacific Design Automation Conference*, 2011.

[24] R. B. Tremaine et al. IBM memory expansion technology (MXT). *IBM J. of Research and Development*, 2001.

[25] G. Wu et al. CAR: Securing PCM main memory system with cache address remapping. In *Proc. Intl. Conference on Parallel and Distributed Systems*, 2012.

[26] B.-D. Yang et al. A low power phase-change random access memory using a data-comparison write scheme. In *Proc. Intl. Symposium on Circuits and Systems*, 2007.

[27] Jun Yang. *Frequent Value Locality and its Applications to Energy Efficient Memory Design*. PhD thesis, University of Arizona, 2002.

[28] Jun Yang et al. Frequent value compression in data caches. In *Proc. Intl. Symposium on Microarchitecture*, 2000.

[29] Jun Yang et al. Frequent value encoding for low power data buses. *ACM Trans. Des. Autom. Electron. Syst.*, 2004.

[30] Jun Yang and R. Gupta. Energy efficient frequent value data cache design. In *Proc. Intl. Symposium on Microarchitecture*, 2002.

[31] Keun Soo Yim et al. Performance analysis of on-chip cache and main memory compression systems for high-end parallel computers. In *Proc of the Intl. Conference on Parallel and Distributed Processing Techniques and Applications*, 2004.

[32] J. Yue and Y. Zhu. Making write less blocking for read accesses in phase change memory. In *Proc. Intl. Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, 2012.

[33] J. Yue and Y. Zhu. Accelerating write by exploiting PCM asymmetries. In *Proc. Intl. Symposium on High-performance Computer Architecture*, 2013.

[34] Youtao Zhang et al. Frequent value locality and value-centric data cache design. In *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[35] P. Zhou et al. A durable and energy efficient main memory using phase change memory technology. In *Proc. Intl. Symposium on Computer Architecture*, 2009.

[36] P. Zhou et al. Energy reduction for STT-RAM using early write termination. In *Proc. Intl. Conference on Computer-Aided Design*, 2009.

[37] Ping Zhou et al. Frequent value compression in packet-based NoC architectures. In *Proc. Asia and South Pacific Design Automation Conference*, 2009.