LOW ENERGY SOLUTIONS FOR MULTI– AND TRIPLE–LEVEL CELL NON-VOLATILE MEMORIES

by

Ali Alsuwaiyan

B.Sc. in Computer Engineering, King Fahd University, 1998

M.Sc. in Computer Engineering, King Fahd University, 2002

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Ali Alsuwaiyan

It was defended on

May 11, 2017

and approved by

Kartik Mohanram, PhD, Associate Professor

Department of Electrical and Computer Engineering, University of Pittsburgh

Hai (Helen) Li, PhD, Associate Professor

Department of Electrical and Computer Engineering, Duke University

Zhi-Hong Mao, PhD,

Department of Electrical and Computer Engineering, University of Pittsburgh

Natasa Miskov-Zivanov, PhD, Assistant Professor

Department of Electrical and Computer Engineering, University of Pittsburgh

Youtao Zhang, PhD, Associate Professor

Computer Science Department, University of Pittsburgh

Dissertation Director: Kartik Mohanram, PhD, Associate Professor

Department of Electrical and Computer Engineering, University of Pittsburgh

LOW ENERGY SOLUTIONS FOR MULTI– AND TRIPLE–LEVEL CELL NON-VOLATILE MEMORIES

Ali Alsuwaiyan, PhD

University of Pittsburgh, 2017

Due to the high refresh power and scalability issues of DRAM, non-volatile memories (NVM) such as phase change memory (PCM) and resistive RAM (RRAM) are being actively investigated as viable replacements of DRAM. However, although these NVMs are more scalable than DRAM, they have shortcomings such as higher write energy and lower endurance. Further, the increased capacity of multi- and triple-level cells (MLC/TLC) in these NVM technologies comes at the cost of even higher write energies and lower endurance attributed to the MLC/TLC program-and-verify (P&V) techniques.

This dissertation makes the following contributions to address the high write energy associated with MLC/TLC NVMs. First, we describe MFNW, a Flip-N-Write encoding that effectively reduces the write energy and improves the endurance of MLC NVMs. MFNW encodes an MLC/TLC word into a number of codewords and selects the one resulting in lowest write energy. Second, we present another encoding solution that is based on perfect knowledge frequent value encoding (FVE). This encoding technique leverages machine learning to cluster a set of general-purpose applications according to their frequency profiles and generates a dedicated offline FVE for every cluster to maximize energy reduction across a broad spectrum of applications. Whereas the proposed encodings are used as an add-on layer on top of the MLC/TLC P&V solutions, the third contribution is a low latency, low energy P&V (L^3EP) approach for MLC/TLC PCM. The primary motivation of L^3EP is to fix the problem from its origin by crafting a higher speed programming algorithm. A reduction in write latency implies a reduction in write energy as well as an improvement in cell endurance.

Directions for future research include the integration and evaluation of a software-based hybrid encoding mechanism for MLC/TLC NVMs; this is a page-level encoding that employs a DRAM cache for coding/decoding purposes. The main challenges include how the cache block replacement algorithm can easily access the page-level auxiliary cells to encode the cache block correctly. In summary, this work presents multiple solutions to address major challenges of MLC/TLC NVMs, including write latency, write energy, and cell endurance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The performance of a computer system heavily depends on the memory subsystem, especially DRAM. Multicore CPUs and parallel processing enable another performance-boosting dimension; however, this performance is bottlenecked by DRAM scalability issue. The issue is due to the fact that DRAM stores bits using capacitors, and DRAM industry faces difficulties in down-scaling DRAM capacitors below 22 nm, due to considerable static and dynamic power requirements [ITRS 2011]. Whereas ideas have been proposed to mitigate this scalability issue of DRAM, e.g., [Mutlu 2013, Liu *et al.* 2012], these ideas only defer the inevitable fact that pure DRAM-based computer systems are going to hit the memory wall.

The continuous demand for higher performance systems has led to extensive research to find a more scalable memory technology, e.g., phase change memory (PCM) [Lee *et al.* 2009] and resistive RAM (RRAM) [Baek *et al.* 2004]. In comparison to DRAM, these technologies are resistance-based, in which logic values correspond to the value of the resistance of the cell. This property makes them more scalable, have an extended data retention time (i.e., non-volatility), and consume only negligible static power.

Even though PCM and RRAM are more scalable than DRAM, they have their own shortcomings too. First, they require higher read latencies and energies than traditional DRAM. Second, they require much higher write latencies and energies in comparison to DRAM. Third, they suffer from a limited cell endurance. These shortcomings must be addressed to make PCM and RRAM viable replacements for DRAM.

Being resistance-based memories, and due to the huge gap between on- and off-resistances, these NVMs can be utilized to store more than one logical bit in one physical cell. This is referred to multi/triple-level cell (MLC/TLC) NVMs in the literature [Nirschl *et al.* 2007, Kang *et al.* 2008, Xu *et al.* 2013]. In this work, we assume that MLC refers to a physical cell that can store 2

logical bits, and TLC refers a cell that can store 3 logical bits. While MLC/TLC enable realizing density and cost advantages, they also aggravate the problems of these NVM technologies, namely, energy, latency, and endurance.

Data encoding solutions (e.g. [Li and Mohanram 2014, Dgien *et al.* 2014, Mirhoseini *et al.* 2012]) try to reduce write energy in NVM technologies. They also implicitly improve NVM endurance [Wang *et al.* 2011], and (in some cases) improve access latency [Niu *et al.* 2013]. Frequent value encoding (FVE) was originally introduced to encode data and address buses [Yang and Gupta 2002, Yang *et al.* 2004, Suresh *et al.* 2009]. FVE can achieve high reductions in NVM write energy by mapping frequent values (words) into low energy codewords. Recently, FVE has been proposed to encode SLC memory words in PCM [Sun *et al.* 2011]. The authors investigated both static (offline) encoding and dynamic (online) encoding. They report that although offline encoding (originally introduced in [Yang and Gupta 2002] by the name "find-once for a given program") achieves higher energy reductions on average, it requires support from compilers and operating system, and therefore, it is infeasible. On the other hand, online encoding provides good energy savings, but requires a non-trivial online profiling effort and therefore affects the overall system performance.

1.1 CONTRIBUTIONS

This work presents three main ideas to mitigate the shortcomings of NVMs:

1. We present an MLC/TLC version of the Flip-N-Write encoding referred to as MFNW [Alsuwaiyan and Mohanram 2015] with the main objective of reducing write energy.

2. We introduce an MLC/TLC version of the FVE encoding that employs machine learning to find perfect codes [Alsuwaiyan and Mohanram 2016] to reduce write energy.

3. We propose low latency, low energy program-and-verify (P&V) approach, referred to as L^3EP, that simultaneously improves MLC/TLC PCM endurance and significantly reduces its write latency and energy in comparison to state-of-the-art P&V for MLC/TLC PCM.

The following subsections present high level summaries of these ideas and break them down by finer grain contributions.

1.1.1 MFNW: An MLC/TLC Flip-N-Write Architecture

The first idea introduces a Flip-N-Write algorithm that is explicitly tailored for MLC NVMs (MFNW). To the best of our knowledge, this is the first work that generalizes the original Flip-N-Write (FNW) algorithm for MLC NVMs. The key idea is the use of cell inversions in place of bit flipping to evaluate encoded forms of the new word for energy reduction.

Below is a break down of the contributions related to MFNW:

- We introduce and investigate two possible variations of the MFNW algorithm: cell Hamming distance (CHD) MFNW and energy Hamming distance (EHD) MFNW. We show that EHD MFNW is more effective in write energy reduction (Section 3.1.3).
- We develop an approximate probabilistic model to facilitate the theoretical analysis of MFNW. This includes the derivation of a closed form expression for the expected number of cell writes in CHD MFNW (Eq. 3.2 in Section 3.1.2). This expression helps determining the optimal word length that maximizes write energy reduction subject to memory overhead constraints. Simulation results show that the closed form expression incurs a negligible error of 1.4% for the chosen sample of word lengths.
- We introduce and evaluate TFNW, the TLC version of MFNW, for TLC NVMs (Section 3.1.4).
- We derive MFNW2 and MFNW3 to further reduce write energy beyond MFNW (Section 3.1.5).
- We propose a cost-aware endurance evaluation that accurately models MFNW/TNFW effects on NVM lifetime in comparison to state-of-the-art MLC/TLC solutions (Section 3.1.6).

Evaluation: Even though MFNW and TFNW are independent of the choice of NVM technology, we evaluate them for the two-bits-per-cell MLC PCM prototype and the three-bits-per-cell TLC RRAM proposed in [Bedeschi *et al.* 2009] and [Xu *et al.* 2013], respectively. We estimate the hardware and delay overheads required to implement MFNW and TFNW for these prototypes. Also, we compare the endurance and average write energy of MFNW with state-of-the-art MLC PCM encoding solutions: data comparison write (DCW) [Yang *et al.* 2007], the energy efficient encoding (EEE) in [Wang *et al.* 2011], and the two-to-three (TTT) encoding in [Mirhoseini *et al.*

2015]. The same comparison is also carried out for TLC RRAM between TFNW and incomplete data mapping (IDM) encoding [Niu *et al.* 2013].

Results: We evaluate MFNW for both MLC PCM and TLC RRAM. MLC PCM results indicate that MFNW achieves up to 39% energy saving over DCW; with MFNW3, the energy saving can be as much as 47% over DCW. Using the proposed endurance evaluation, MFNW, without using any wear-leveling or error-correction techniques, can prolong memory lifetime by up to 100% in comparison to DCW. For TLC RRAM results, TFNW achieves up to 53% energy saving over DCW and, without using any wear-leveling or error-correction techniques, it is capable of extending the lifetime of RRAM NVM by up to 87% in comparison to DCW. Detailed findings and comparisons with state-of-the-art encoding solutions are presented in Section 3.2.

1.1.2 Frequent Value Encoding

The second idea presents an offline frequent value encoding (FVE) for MLC/TLC NVMs that achieves an average write energy reduction approaching, and sometimes exceeding, that of optimal offline encoding [Yang and Gupta 2002]. The proposed method, which does not require compiler or operating system support, is based on finding multiple optimal FVEs. Each FVE is derived by aggregating the data frequency profiles of a group of compatible applications. To the best of our knowledge, this is the first work that presents a feasible version of an offline FVE approaching the average energy reductions of optimal offline FVE.

Below is a break down of the contributions related to the proposed FVE solution:

- We describe the use of k-medoids algorithm [Park and Jun 2009] to cluster a set of general-purpose benchmark applications (SPEC CPU2006 [SPEC CPU 2006]) into k groups, in which each group contains compatible set of applications. The applications are represented as an observation matrix consisting of rows of feature sets that are essential inputs to the k-medoids algorithm (Section 4.1.1).
- We process the output of the k-medoids algorithm to produce offline, low overhead, energy-efficient FVEs. The objective is to reach the write energy of optimal offline FVE while avoiding its disadvantages (Section 4.1.2). The proposed codec (coder/decoder) architecture uses read-

only memories (ROMs) that are known to consume low power/energy in comparison to lookup tables (LUTs) and content-addressable memories (CAMs) (Section 4.1.3).

Evaluation: For a realistic evaluation of the proposed FVE solution, we divide the set of benchmark applications into training and evaluation sets. The training set is used to derive the k offline FVE mappings, which are then used to encode the applications in the evaluation set. The energy saving of our solution across the evaluation set is an accurate indicator of its quality against other new applications that are not considered in this work. However, the training set should accurately model the class of applications to be expected. For example, in this work, the training set models a broad set of general-purpose applications (SPEC benchmarks).

Results: We evaluate the proposed FVE solution for both MLC PCM and TLC RRAM. MLC PCM results indicate that average write energy is only 5% more than that of the optimal offline FVE. The result is even better in the TLC RRAM case, where the average write energy of the proposed technique is 1% less than that of the optimal offline FVE. In comparison to the write energies of the MLC and TLC DCW [Yang *et al.* 2007], the proposed solution achieves 39% and 35% energy savings, respectively, while the memory overhead in both cases for our solution does not exceed 3.5%. We report the results of our proposed solution for different values of k (the number of clusters), and for some values, our method results in average write energy that is only 4% more than that of the state-of-the-art MLC PCM TTT encoding in [Mirhoseini *et al.* 2015]. However, the big difference is that TTT requires 50% NVM overhead (16$\times$ more than ours) for the case study presented in that paper. While the 50% overhead of TTT can be reduced significantly, we argue that the reduction will result in exponentially larger sizes of LUTs than the sizes of our codec ROMs. For TLC RRAM, our method results in the same average write energy as IDM [Niu *et al.* 2013]. However, IDM has a memory overhead of 20%, i.e., 5.7$\times$ more than the proposed work. Detailed findings and comparisons with state-of-the-art encoding solutions are presented in Section 4.2.

1.1.3 L^{3}EP

Whereas encoding schemes try to alleviate the side effects of P&V approaches, our third idea is a low latency, low energy (L^{3}EP) P&V approach for MLC/TLC PCM. L^{3}EP simultaneously reduces the write latency and energy, and improves the endurance of PCM cells.

Below is a break down of the contributions related to L^{3}EP:

- L^{3}EP utilizes a multiple linear regression model to reach the target TLC state in just one (at most five) pulse(s) for 53% (>95%) of our comprehensive Monte Carlo (MC) simulations.
- L^{3}EP also accelerates the naturally slow PCM crystallization process by packing crystallization pulses separated by short idle periods, and augmenting them with a single verify step. L^{3}EP thus avoids unnecessary P&V verification steps resulting in faster crystallization over state-of-the-art P&V approaches.
- We describe a comprehensive framework to compute critical L^{3}EP P&V-related parameters for multiple technology nodes (Section 5.1.2). The framework uses a series of MC simulations and optimizations imposing bounds on those parameters for the target technology node. Although this work focuses on TLC PCM, it can also be applied to MLC PCM through appropriate write margin and resistance ranges parameterization, as explained in Section 5.1.2.

Evaluation: We compare L^{3}EP to three state-of-the-art TLC PCM P&V approaches: (i) staircase up programming (SCUP) [Bedeschi *et al.* 2009], (ii) proportional-integral-derivative programming (PIDP) [Papandreou *et al.* 2011], and (iii) dual-pulse programming [He *et al.* 2014] (an amorphization-only P&V approach referred to as AOP henceforth). The comparisons are performed at both the cell-level as well as full system level.

Results: Results at the cell-level indicate that L^{3}EP can reduce the mean latency (energy) by 2.4–15× (1.9–12.2×) in comparison to SCUP, PIDP, and AOP. Moreover, L^{3}EP reduces the worst case latency (energy) by 2.8–9.1× (2.1–11.4×) in comparison to SCUP, PIDP, and AOP. Despite this significant improvement in the programming latency, energy, and endurance of TLC PCM, the hardware overhead of L^{3}EP is comparable to SCUP, PIDP, and AOP.

Full system simulation results indicate that the average access latency and total energy of L^{3}EP are 2.1–8.3× and 1.6–3.7× lower, respectively, in comparison to SCUP, PIDP, and AOP. Further-

more, the IPC and memory bandwidth of L^{3}EP are 1.6–5$\times$ and 1.6–4.6$\times$ higher, respectively, in comparison to SCUP, PIDP, and AOP. Detailed findings and comparisons with state-of-the-art P&V solutions are presented in Section 5.2.

1.2 THESIS ORGANIZATION

The rest of this document is organized as follows. Chapter 2 covers basic concepts of NVMs and related encoding and P&V solutions for MLC/TLC NVMs. Chapter 3 presents the theory and results of the proposed MFNW encoding solution. Chapter 4 introduces our proposed FVE solution for MLC/TLC NVMs. L^{3}EP is presented in Chapter 5. Finally, directions for future research are summarized in Chapter 6.

2.0 BACKGROUND

This chapter is divided into three sections. The first two sections cover background material related to the proposed MFNW and FVE schemes. The third section is background material for the proposed TLC program-and-verify (P&V) approach, and it overviews concepts related to MLC/TLC P&V approaches of MLC/TLC PCM.

2.1 BASICS OF NVMS

A PCM cell is made of chalcogenide material which has two states, amorphous and crystalline corresponding with high and low resistances, respectively. To program logic 0 in a PCM cell, current is applied to heat the cell above its melting point followed by a quick cool down. Programming logic 1 is similar, but the target heating temperature is lower, and the rate of cooling is slower; as a result programming logic 1 takes more time [Lee *et al.* 2008, Kang *et al.* 2011].

An RRAM cell is composed of three layers: two metal electrode layers surrounding a transition metal-oxide layer (MOL). The MOL can be switched between high and low resistance states by applying set and reset voltages, respectively. The change in the state of the MOL is a result of forming and deforming the conductive filaments (CF). When Oxygen holes line up from the top to the bottom electrode, a CF is formed, and current can pass through the cell, resulting in a low resistance MOL. Applying a reset voltage disrupts the CF and results in a high resistance MOL [Wong *et al.* 2012].

While these NVM technologies offer extended data retention times, high scalability, and very low static power consumption, they suffer from shorter lifetime, high cell programming energy, and high read/write latencies in comparison to DRAM. For example, STT-RAM requires about

$10\times$ more energy than DRAM per write access [Chang *et al.* 2013]. These problems are worse in MLC NVMs, where a memory cell stores more than one logical bit.

Multi-level cell NVM: While every physical cell in SLC NVMs stores one logical bit of information, a physical cell in MLC NVMs can store more than one logical bit (typically 2 and 3). Each level in a multi-level cell encodes a state, and 2^m states can be encoded using m bits per cell. Program-and-verify (P&V) [Bedeschi *et al.* 2009] is commonly used to encode MLC states in PCM and RRAM. To program a cell to a specific state, P&V loops through a program phase followed by verify phase until the target state is reached. The all–0 and all–1 states are the easiest to program in terms of write energy. The further a state from these boundary states, the more P&V loops are required, i.e., the write energy of these non-terminal states is higher. In other words, write energy peaks as we get closer to the middle state. Due to the high variability of the manufacturing of PCM and RRAM arrays, the number of iterations required to program a certain state is unknown, and programming happens slowly and gradually until we are within some acceptable tolerance of the destination state.

2.2 REVIEW OF ENCODING SOLUTIONS

Data comparison write (DCW) [Yang *et al.* 2007]: DCW is also referred to as read-modify-write, and it is the simplest way to avoid unnecessary bit writes in NVMs. Whenever there is a new word to be written to memory, DCW only writes the bits that are different. For 2^m–state MLC memory, assuming each state has 2^{-m} probability of occurrence, a write event has a probability of $1-2^{-m}$. It follows from the Binomial distribution theorem that the expected number of cell writes in DCW is $n \cdot (1-2^{-m})$, where n is the number of cells per word.

Flip-N-Write: FNW for SLC memories encodes data with the objective of minimizing the number of bit writes. Each memory word is associated with a tag bit, which is 0 if the word is stored as-is, or 1 if the word is stored in bit-wise complemented form. When a write request arrives at the memory controller, FNW writes the new word as-is if the Hamming distance between the new and old words is not greater than $n/2$, where n is the number of bits per word (excluding the tag bit).

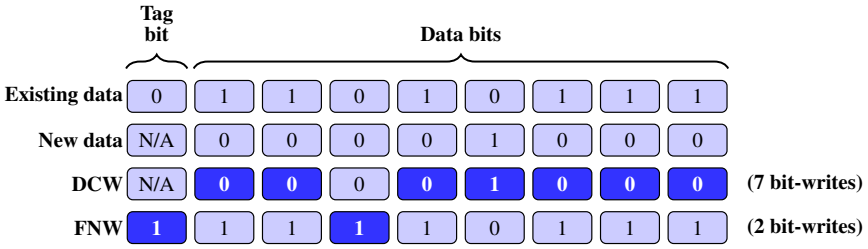

Figure 2.1: For the shown existing and new data, DCW results in 7 bit-writes, whereas FNW, results in only 2 bit-writes. Bit-writes are indicated with darker-color backgrounds.

If the Hamming distance is greater than $n/2$, FNW writes the new word in bit-wise complemented form. While it can be easily shown that the peak number of bit-writes in FNW cannot exceed $\lceil n/2 \rceil$, the expected number of bit-writes, as reported in [Cho and Lee 2009], is

$$\frac{1}{2^{n+1}} \left(\sum_{k=0}^{n/2} k \binom{n+1}{k} + \sum_{k=n/2+1}^{n+1} (n+1-k) \binom{n+1}{k} \right)$$

Fig. 2.1 illustrates the potential advantage of FNW over DCW. While DCW results in 7 bit-writes, FNW uses the complemented version of the new word and results in only 2 bit-writes (71% improvement).

FNW interestingly alleviates all three problems of NVMs simultaneously. First, since it reduces the average number of bit writes per memory word, it reduces the average energy consumption per word. Second, the reduction of the expected number of bit writes per word, as well as bounding the peak number of bit writes per word, definitely improves memory endurance and prolongs lifetime, since the probability of writing to the same memory cell is reduced. Third, improving the write latency is a direct result of the upper bound on the number of bit writes per word, implying that we can write at least two words at a time without violating the energy consumption limits in modern NVM memory module controllers [Yue and Zhu 2012]. For these reasons, we chose to design and implement an MLC version of FNW, believing that it will help alleviate the problems in MLC NVMs as well.

FlipMin [Jacobvitz *et al.* 2013] is a variation of SLC FNW that utilizes coset coding to generate more encoded versions of the new word and reduce the number of bit flips. This enables FlipMin to achieve more energy reduction than FNW at the cost of k overhead bits in comparison to one

bit overhead for FNW. FlipMin is a generalization of FNW, and it behaves identical to FNW when $k = 1$.

PRES [Seyedzadeh *et al.* 2015] is another coset based write minimization scheme that encodes the new word into a set of random codewords, which outperforms both FNW and FlipMin. The authors demonstrate (by simulation and mathematically) that random codewords result in less number of flips in comparison to FNW and FlipMin. Therefore, they derive a reversible pseudo-random encoding scheme and prove its randomness by random code tests. PRES has the same NVM overhead as FlipMin.

CAFO [Maddah *et al.* 2015] is another variation of SLC FNW that slices the new word into a 2-dimensional $r \times c$ matrix and performs row- and column-wise bit flipping operations. It aims at minimizing the write cost, e.g., write energy and read reliability in PCM and spin-transfer torque random access memory (STT-RAM), respectively. CAFO achieves more write cost reduction than FNW. This is due to using a heuristic to identify unnecessary flips that is usually overlooked in FNW. Also, incurring more NVM overhead in comparison to FNW helps achieving further cost reduction. The excess overhead of CAFO in comparison to FNW is $\min(r, c)$ bits.

Offline frequent value encoding (FVE) [Yang and Gupta 2002]: Motivated by the fact that most computer applications have high data locality, FVE encodes frequent values such that they are assigned to the least energy codewords. It has been reported that these frequent values may account for 32% of all the values written to memory by a specific application [Yang *et al.* 2004]. Consider an example where an application has only four possible values: A, B, C, and D:

Value	A	B	C	D
Frequency	20	5	10	4
Energy (pJ)	36	307	547	20

In this table, the 1st row is the possible value, 2nd is the corresponding frequency of occurrence of that value, and 3rd is the corresponding write energy for each value. Therefore, the total energy consumed by this application is 7,805 pJ. The following value-codeword assignment:

Value	A	C	B	D
Codeword	D	A	B	C

reduces the total energy to 4,483 pJ, i.e., 43%. The assignment ranks the values in decreasing frequencies and codewords in increasing energies, and assigns values with high frequency to code-

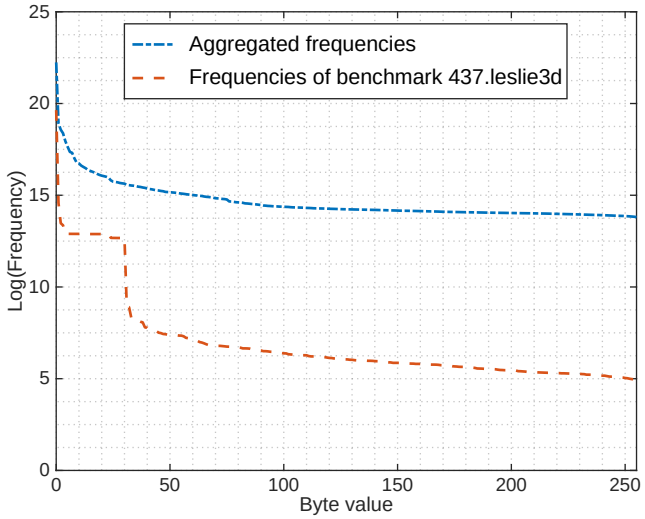

Figure 2.2: In this plot, the x-axis is all the possible values (bytes) and the y-axis is the frequency of the corresponding value in logarithmic scale. Clearly, the frequency profile of benchmark 437.leslie3d exhibits good skew that supports the locality of frequent values. But aggregating the frequency profiles of 32 benchmarks results in a relatively flat plot, suggesting that more than 98% of the values are almost equally likely.

words with low energy in the order of the ranking. For example, A has the highest frequency and D has the lowest energy, we assign value A the codeword D. This optimal offline code assignment will only be good if used for this specific application (or a similar one), and therefore it is referred to as *optimal perfect knowledge* assignment, i.e., find-once for a given program [Yang and Gupta 2002]. Henceforth, we simply refer to this method as the perfect knowledge method.

In practice, if we have more than one application, we aggregate their frequency profiles and generate a combined code assignment resulting in lower energy. However, combining frequencies of incompatible applications may eliminate skew that is essential to achieve high energy reductions, as illustrated in Fig. 2.2. The plots show (1) the aggregate frequency plot of 32 distinct benchmark applications (28 SPEC CPU2006 [SPEC CPU 2006] and four Splash-2 [Arnold *et al.* 1992]) and (2) the frequency plot of the leslie3d benchmark. Even though most offline frequency profiles of individual applications are skewed in favor of FVE, aggregating frequency profiles may eliminate skew and plateaus the graph for almost 98% of possible values. This is in comparison to the individual frequency plot of the benchmark leslie3d, in which the last elbow in the plot appears after 23% of possible values, implying high energy reduction.

Most applications have skewed frequency profiles in favor of perfect knowledge FVE. A primary motivation for this work is the careful aggregation of the frequency profiles of applications to derive multiple encodings, in which each encoding is still a result of a skewed aggregate frequency profile.

The authors in [Sun *et al.* 2011] evaluated perfect knowledge FVE on SLC PCM. While they reported that average write energy is greatly reduced, perfect knowledge FVE requires profiling support during the compilation process, implying a long compilation time for most applications. Another disadvantage of perfect knowledge FVE is that it requires operating system support, since for every application, we need to store the code assignment specific to that application in its executable. Perfect knowledge FVE is therefore expensive in practice.

On the other hand, online profiling occurs during application execution, e.g., [Sun *et al.* 2011]. Frequent values are identified on the fly and may also be replaced by other more frequent values. Although online profiling provides adaptive write energy reduction, it requires significant hardware and may affect overall system performance.

Encoding solutions for MLC/TLC NVMs: Besides DCW, FNW, FlipMin, PRES, and CAFO, many data encoding solutions have been proposed for SLC NVMs, e.g. [Li and Mohanram 2014, Dgien *et al.* 2014, Mirhoseini *et al.* 2012, Sun *et al.* 2011, Cho and Lee 2009]. Most of SLC encoding solutions can be readily applied to MLC/TLC NVMs with minor and sometimes no changes. In the following few paragraphs, we summarize the main ideas of some encoding solutions that have been designed specifically for MLC/TLC NVMs.

Zero-value-based morphable PCM (ZM-PCM) [Arjomand *et al.* 2011] is an MLC PCM main memory architecture design that is based on and motivated by the observation that the zero value frequently occurs in PARSEC-2 benchmarks. Driven by this motivation, ZM-PCM encodes these zero-valued MLC cells into SLC resistance levels, implying improved latency, energy, and lifetime. The key element is the use of morphable cells, in which the number of logical bits per physical cell can range from 1 to n (in practice, n cannot exceed six under the utilized P&V algorithm). Therefore, ZM-PCM requires additional memory overhead to track the number of logical bits stored in each physical cell.

In [Wang *et al.* 2011], a cell-level mapping has been proposed to reduce write energy in MLC PCM. The idea is to use a set of six mappings, in which one mapping type is chosen to be applied

to a PCM line at a time, depending on the statistics of the states of the cells in that line. The objective of these mappings is to reduce the number of high power state (HPS) cells and increase the low power state cells (LPS). The mapping type of a PCM line is tracked by using a tag value composed of two cells. A cell state will only contribute to the line statistics if it results in a write operation under DCW. In Section 3.2, we refer to this encoding as the energy efficient encoding (EEE).

Write truncation (WT) [Jiang *et al.* 2012b] dynamically identifies the PCM cells that require more P&V iterations than other cells in the memory line, and truncates their last several iterations to match a preset maximum number of iterations. The objective is to finalize the PCM write early, and therefore improve latency and reduce write energy. WT is integrated with an error correction code (ECC) to recover truncation errors. The same reference proposes "form switch" (FS) to mitigates the additional storage overhead required by ECC. FS utilizes frequent pattern compression (FPC) [Alameldeen and Wood 2004] to reduce storage overhead requirement. Moreover, FS will store a PCM line in SLC format if it can be compressed to half its size or less. This implies shorter access latency and better write endurance.

Elastic RESET (ER) [Jiang *et al.* 2012a] proposes to use only 3 states of the 4-state MLC PCM to reduce MLC reset current, and therefore reduce write energy and improve endurance. ER first compresses the memory line using FPC, and an encoding scheme is chosen based on the the compression ratio. If the compression ratio is 50% or less, data is stored in SLC form [Arjomand *et al.* 2011]. If the ratio is between 50% and 75%, fraction encoding is employed, in which three logical bits are mapped into two physical cells. If the ratio is more than 75%, the cells are stored in regular MLC form. Two SLC bits are associated with every PCM line to distinguish these cases during decoding.

Incomplete data mapping (IDM) [Niu *et al.* 2013] has been proposed to encode TLC RRAM. In IDM, a TLC memory line is converted into another form, in which the cells are 6-state cells. The purpose of this conversion is to eliminate the two high energy and latency states. This encoding is achieved by mapping every chunk of 5 logical bits in the memory line into 6 logical bits (two physical cells), while restricting the use of encoded states to the lowest six states (i.e., the lowest with respect to the state write energy). IDM reduces both write energy and latency, but incurs a memory overhead of 20%.

In [Mirhoseini *et al.* 2015], an encoding scheme has been proposed for MLC PCM. This method assigns $(n+1)$-cell codeword to n-cell word. The word may be any value in the range from 0 to $4^n - 1$. The 4^n corresponding codewords are selected from the range from 0 to $4^{n+1} - 1$ such that the number of intermediate high-energy states (01 and 10) is minimized. An example code-assignment table is given in [Mirhoseini *et al.* 2015] for the case of $n = 2$. For this code-assignment, the expected write energy reduction is about 40%. However, this is at the cost of 50% memory overhead. As n increases, we expect more saving in energy; however, the code-assignment table grows exponentially with n. We compare MFNW with this encoding scheme for the case $n = 2$, assuming the same NVM overhead for MFNW. In our comparison charts, we refer to the case of $n = 2$ as two-to-three (TTT) encoding.

In [Palangappa and Mohanram 2016], a compression-expansion (CompEx) coding has been proposed. CompEx combines compression and expansion coding to realize energy reduction in MLC/TLC NVMs. The authors evaluated CompEx using both frequent pattern compression (FPC) [Alameldeen and Wood 2004] and base-delta-immediate (BΔI) compression [Pekhimenko *et al.* 2012]. The compression step is followed by an expansion step to avoid writing high energy states.

2.3 P&V RELATED BACKGROUND

This section covers relevant concepts and provides an overview of state-of-the-art P&V solutions for MLC/TLC PCM.

Multi–/Triple–level cell (MLC/TLC): An MLC (TLC) can store two (three) logical bits corresponding to four (eight) distinct MLC (TLC) states. The encoding of MLC/TLC states is achieved by controlling the resistance level of the PCM cell, i.e., each MLC/TLC state has its own unique resistance range that does not overlap with other states. However, instead of maintaining lower and upper bounds of resistance ranges for every MLC/TLC state, it is more efficient to assign a mid-point resistance (R_{M_i}) of the resistance range corresponding to the i^{th} MLC/TLC state, and a write margin, ϵ, to define a mapping from a PCM cell resistance, R, to an MLC/TLC state. In other words, a PCM cell is assigned to the i^{th} MLC/TLC state if $|R_{M_i} - R| \leq \epsilon$.

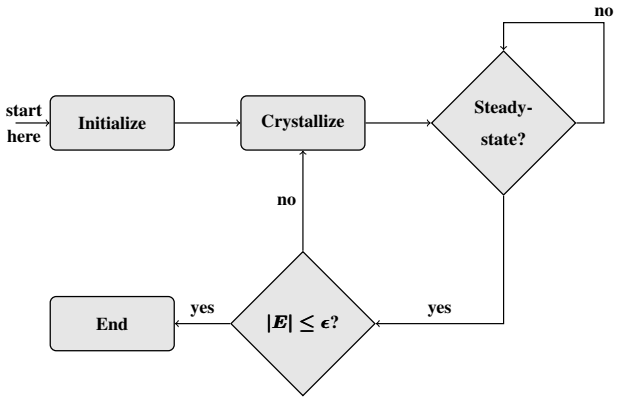

Figure 2.3: In SCUP P&V [Bedeschi *et al.* 2009], SCUP first initializes the cell to the amorphous state. Next, SCUP issues a partial crystallization pulse to slightly lower the resistance, waits for the steady-state resistance, and verifies if the programming error ($|\boldsymbol{E}|$) is within the write margin (ϵ). If so, SCUP stops, otherwise it repeats the process until the target resistance is reached ($|\boldsymbol{E}| \leq \epsilon$).

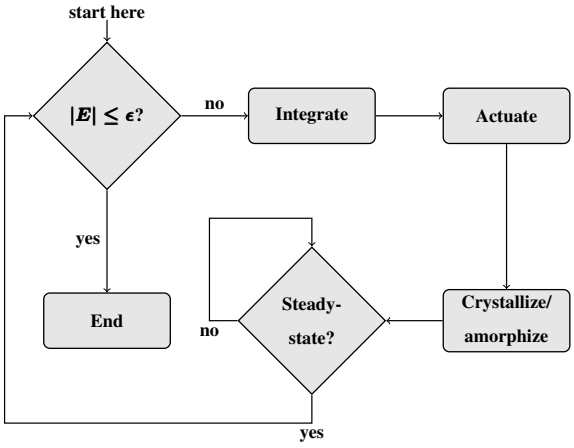

Figure 2.4: In PIDP P&V [Papandreou *et al.* 2011], PIDP examines if $|\boldsymbol{E}| \leq \epsilon$ and stops if true. Otherwise, it integrates the error signal, converts it into a control command (actuation), and issues either a crystallization or amorphization pulse, depending on the outcome of the actuation. Next, it waits for the steady-state resistance and repeats this process until $|\boldsymbol{E}| \leq \epsilon$.

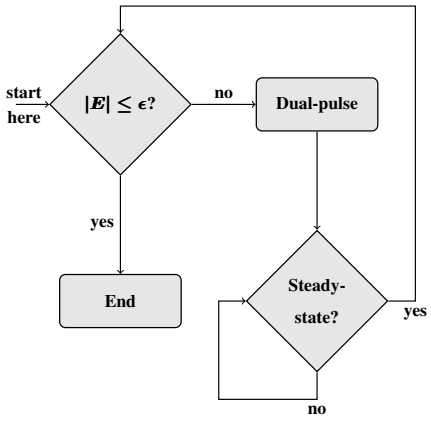

Figure 2.5: In AOP P&V [He *et al.* 2014], AOP examines if $|E| \leq \epsilon$ and stops if true. Otherwise, based on the error, AOP guesses the amplitude of the first dual-pulse using a simple linear model. Next, it waits for the steady-state resistance and repeats this process until $|E| \leq \epsilon$. Subsequent amplitudes of the dual-pulses are slight increments (or decrements) of the previous amplitude.

PCM P&V strategies: The two P&V strategies are crystallization-based and amorphization-based. In the crystallization-based strategy, the cell is initialized to the highest resistance state using a full RESET pulse. The PCM resistance is then gradually reduced by applying short crystallization pulses with increasing amplitudes, until the target MLC/TLC state is reached. In the amorphization-based strategy, the cell is initialized to the lowest resistance state using a full SET pulse. The PCM resistance is then gradually increased by applying short amorphization pulses with increasing amplitudes, until the target MLC/TLC state is reached. The key difference between the two strategies is the amplitude range of the pulses, i.e., crystallization pulses have smaller amplitudes than amorphization pulses [Burr *et al.* 2010, Braga *et al.* 2010, Bedeschi *et al.* 2009, He *et al.* 2014].

PCM model: Our proposed P&V approach (L^3EP) leverages an accurate state-of-the-art compact model for a PCM cell [Xu *et al.* 2012] to support a comprehensive analysis of the effects of manufacturing variability on write latency and energy in both the crystallization-based and amorphization-based strategies. This device-level model can capture the underlying physical mechanisms of phase change and the dependence on material/structure parameters. The physical nature of this model further helps incorporate variability and reliability issues in the analysis. Note that L^3EP uses this model primarily to simulate the phase change behavior of the PCM cell;

readout operations are modeled by the maximum duration of the PCM cell readout. This tradeoff is essential for an accurate modeling of phase change to simulate the various P&V approaches. In this work, when a P&V approach requires sampling the PCM cell resistance, we assume that the sampling time is ≤ 10 ns [Burr *et al.* 2010].

State-of-the-art P&V: There are three state-of-the-art P&V approaches: staircase up programming (SCUP) [Bedeschi *et al.* 2009], proportional-integral-derivative programming (PIDP) [Papandreou *et al.* 2011], and amorphization-only P&V (AOP) [He *et al.* 2014]. Note that we exclude the pulse-tail slope-tuning P&V [Nirschl *et al.* 2007] from our comparisons, since it has a longer latency due to the accumulation of the long tails of its programming pulses. Fig. 2.3 shows the flowchart of SCUP. SCUP starts by applying a full RESET pulse causing the phase change material ($Ge_2Sb_2Te_5$, or GST for short) to transform to the full amorphous state. This is followed by successive P&V iterations that incrementally crystallize the GST until the absolute value of the programming error, $|E|$, is $\leq \epsilon$, the predefined write margin. The amplitude of the programming pulse starts from the voltage causing the lowest crystallization temperature. In each iteration, the amplitude is incremented by a fixed ΔV. Although larger values of ΔV result in faster programming, they also affect programming reliability, since SCUP uses a crystallization-based P&V strategy with only one programming direction. In practice, ΔV can be optimized using Monte Carlo (MC) simulations.

Whereas SCUP updates the pulse amplitude in fixed increments independently of the programming error, PIDP updates the pulse amplitude dynamically in response to a summarized history of the programming error. PIDP achieves this by using an integrator to accumulate error over time. Fig. 2.4 shows the flowchart of PIDP. The programming loop starts with a verification step, which samples the PCM cell resistance (R) and calculates the error $E = R - R_M$, where R_M is the midpoint resistance of the target TLC state. If $|E| \leq \epsilon$, PIDP stops programming. If $|E| > \epsilon$, PIDP proceeds to the "Integrate" step to sum the error and compute a control command, which is then converted to a pulse amplitude in the "Actuate" step. Subsequently, the programming pulse is issued during the "Crystallize/amorphize" step. Note that PIDP idles until the steady-state resistance is reached and then returns to the verification step to begin a new loop.

In contrast to SCUP and PIDP, which utilize crystallization pulses, the authors in [He *et al.* 2014] describe an amorphization-only P&V (AOP) technique to reach any cell resistance level

by the application of a dual-pulse instead of just a single pulse on every iteration. The authors report that the heat accumulation effect between double pulses with a narrow interval is primarily responsible for the high controllability of the cell resistance. In AOP, any cell resistance level can be reached by controlling three parameters of the dual-pulse: pulse amplitude (V), width (W), and interval (T). In this work, AOP is integrated into a P&V loop by controlling V, while keeping W and T constant. Fig. 2.5 shows the flowchart of AOP. Similar to PIDP, the loop starts with a verification step, which samples the PCM cell resistance (R) and calculates the error $E = R - R_M$. If $|E| \leq \epsilon$, the target TLC state is reached. If $|E| > \epsilon$, AOP predicts V of the first dual-pulse using a simple linear predictor. A dual-pulse of amplitude V is then issued during the "Dual-pulse" step. Finally, AOP waits for the steady-state resistance and then returns to the verification step for a new iteration. From the second iteration onward, AOP updates V in fixed increments or decrements according to the sign of E.

3.0 MFNW: AN MLC/TLC FLIP-N-WRITE ARCHITECTURE

Flip-N-Write (FNW), originally introduced as "bus-invert coding" in [Stan and Burleson 1995], has been proposed to encode single-level-cell (SLC) memory words in PCM [Cho and Lee 2009]. It has the appeal to simultaneously realize improvements in write energy, endurance, and latency. To write a memory word to a given destination address, FNW tracks the number of bit-writes required to overwrite the old word using the new word (i) as-is and (ii) in bit-wise complemented form. FNW chooses the option that results in minimum bit-writes to reduce the write energy per word. For decoding, FNW tracks the chosen alternative using a tag bit that is associated with each memory word. In addition to reducing energy, in the long run, FNW also improves memory endurance since it also reduces the number of bit-writes. Note that FNW has been applied to SLC PCM, where each memory cell stores a single logical bit. In contrast, in MLC NVMs, a single cell can store more than one logical bit realizing density and cost advantages [Nirschl *et al.* 2007, Kang *et al.* 2008, Xu *et al.* 2013]. The increase in the storage capacity per cell comes at the cost of increased programming energy, read latency, as well as lower endurance in comparison to SLC NVMs. Therefore, there is a strong motivation to develop data encoding and wear-leveling techniques for MLC NVMs [Wang *et al.* 2011, Niu *et al.* 2013, Wen *et al.* 2014].

The contributions in this chapter are as follows. First, we introduce MFNW, a Flip-N-Write algorithm explicitly tailored for MLC NVMs (Section 3.1.1). To the best of our knowledge, this is the first work that generalizes the original FNW algorithm for MLC NVMs. The key idea is the use of cell inversions in place of bit flipping to evaluate encoded forms of the new word for energy reduction. Second, we introduce and investigate two possible variations of the MFNW algorithm: cell Hamming distance (CHD) MFNW and energy Hamming distance (EHD) MFNW. We show that EHD MFNW is more effective in write energy reduction (Section 3.1.3) . Third, we develop an approximate probabilistic model to facilitate the theoretical analysis of MFNW.

This includes the derivation of a closed form expression for the expected number of cell writes in CHD MFNW (Eq. 3.2 in Section 3.1.2) . This expression is important because it helps determining the optimal word length that maximizes write energy reduction subject to memory overhead constraints. Simulation results show that the closed form expression incurs a negligible error of 1.4% for the chosen sample of word lengths. Fourth, we introduce and evaluate TFNW, the TLC version of MFNW, for TLC NVMs (Section 3.1.4). Fifth, we propose MFNW2 and MFNW3 that can further reduce energy consumption beyond MFNW (Section 3.1.5). Finally, we introduce a cost-aware endurance evaluation methodology that we use to evaluate MFNW/TNFW effects on NVM endurance in comparison to state-of-the-art MLC/TLC solutions (Section 3.1.6).

Even though MFNW and TFNW are independent of the choice of NVM technology, we evaluate them for the two-bits-per-cell MLC PCM prototype and the three-bits-per-cell TLC RRAM proposed in [Bedeschi *et al.* 2009] and [Xu *et al.* 2013], respectively. We estimate the hardware and delay overheads required to implement MFNW and TFNW for these prototypes. Also, we compare the endurance and average write energy of MFNW with state-of-the-art MLC PCM encoding solutions: data comparison write (DCW) [Yang *et al.* 2007], the energy efficient encoding (EEE) in [Wang *et al.* 2011], and the two-to-three (TTT) encoding in [Mirhoseini *et al.* 2015]. The same comparison is also carried out for TLC RRAM between TFNW and incomplete data mapping (IDM) encoding [Niu *et al.* 2013].

MLC PCM results indicate that MFNW achieves up to 39% energy saving over DCW; with MFNW3, the energy saving can be as much as 47% over DCW. Using the proposed endurance evaluation, MFNW, without using any wear-leveling or error-correction techniques, can prolong memory lifetime by up to 100% in comparison to DCW. For TLC RRAM results, TFNW achieves up to 53% energy saving over DCW and, without using any wear-leveling or error-correction techniques, it is capable of extending the lifetime of RRAM NVM by up to 87% in comparison to DCW. Detailed findings and comparisons with state-of-the-art encoding solutions are presented in section 3.2.

In summary, MFNW3 results in lower write energy, longer endurance than state-of-the-art MLC NVM encodings. On the other hand, whereas TFNW provides a 25% reduction in the number of cell writes over IDM, it only results in comparable energy and endurance to IDM. Intuitively, cell write reduction should improve both energy and endurance; however, in contrast to TFNW,

IDM completely avoids cell writes to the high energy TLC states (States 3 and 4), resulting in the two approaches having comparable energy and endurance.

This chapter is organized as follows. Section 3.1 and 3.2 present the contributions and evaluation results, respectively. Section 3.3 summarizes our findings and concludes the chapter. Finally, Section 3.4 is a supplementary section that lists and proves some observations about MFNW and TFNW.

3.1 CONTRIBUTIONS

This section starts with a high level overview of MFNW, followed by a discussion of the two MFNW modes: CHD and EHD. This is followed by a theoretical analysis to derive a closed-form expression for the expected number of cell writes of CHD MFNW. This expression is then used to derive an upper bound for the expected energy consumption of EHD MFNW. Additionally, we introduce the TLC version of FNW (TFNW) and the other variations of MFNW, which allow further energy reductions (MFNW2 and MFNW3). The section concludes by describing our proposed endurance model that we assume during memory lifetime simulation.

3.1.1 MLC FNW (MFNW)

We begin with the definitions that are necessary to present MFNW. We define the *replication* operator $\{n\{a\}\}$ as in Verilog: replicate the binary string a, n times. For example, $\{4\{01\}\}$ is the binary string 01010101. We also borrow the *concatenation* operator from Verilog, i.e., $\{01, 1100\}$ =011100. Further, we define the i^{th} cell inversion of a multi level cell c as $i \oplus c$, where $\oplus$ is the bit-wise XOR operator, assuming that i and c have the same number of bits. For example, if $c = 01$, then the 0^{th} inversion of c is $00 \oplus 01 = 01$; the 1^{st} inversion is $01 \oplus 01 = 00$, and so on. Note that the number of possible cell inversions of c is 2^m, where m is the width of c in bits.

Inversion operator: Assume that the MLC is m-bit wide, i is a single cell, and a is n-cell wide. We define the i^{th} inversion of a as $\{i, \{n\{i\}\} \oplus a\}$. Note the use of both replication and concatenation operators in this definition. Also, note that the i^{th} inversion of a can be defined equivalently as

i concatenated with the i^{th} cell inversions of every cell in a, starting with the most significant cell. Clearly, the number of possible inversions for a is 2^m. To illustrate the inversion operator, assume MLC NVM, where every cell is 2-bit wide. If $a = 001110$, then the 0^{th} inversion of $a = \{00, 000000 \oplus 001110\} = 00001110$; 1^{st} inversion of a is 01011011, and so on. Note if i is one bit, the 0^{th} inversion of a is a and the 1^{st} inversion of a is $\overline{a}$, i.e., the 1's complement of a.

MFNW write: With this definition of inversion operator, we can generalize the bit-flipping in SLC FNW to cell-inversion in MFNW as follows. Suppose the memory controller receives a write request, with a new word W_2 to replace an old word W_1. Without loss of generality, let us assume that the old word is already fetched from the NVM array into the controller. This means that the old word is 1 cell wider than the new word, because it contains the tag cell. MFNW generates all possible cell inversions of W_2, and computes the distance between these inversions and the old word W_1. The *closest* inversion is chosen to overwrite W_1. Algorithm 1 outlines these steps in an algorithmic manner.

Algorithm 1: MFNW

input : W_1: existing data, W_2: tag-less data to write, n: number of bits/cell, n: number of cells/word
effect : An inversion of W_2 written at address A, which is the address of W_1

1. Compute the i^{th} inversion of W_2, $W_{2,i} = \{i, \{n\{i\}\} \oplus W_2\}$, for $0 \leq i < 2^n$.
2. Find k such that the Hamming distance between W_1 and $W_{2,k}$ is minimum.
3. Write *the closest* inversion ($W_{2,k}$) at address A

The algorithm needs to know the number of bits per cell and the number of cells per word excluding the tag cell. These two parameters are constants while the dynamic parts of input parameters are the old word (including the tag cell) and the new tagless word. The interpretation of the *closest* inversion to the old word can mean the closest with respect to CHD or EHD. We define and illustrate both distance metrics in sub-sections 3.1.2 and 3.1.3.

MFNW read: When the CPU needs to read a memory location, the MFNW memory controller must decode the memory word at that address to recover the original tagless word. This is done by bit-wise-XORing the word with its tag cell replicated n times, assuming the memory word length (without the tag cell) is n cells. For example, assuming an MLC NVM, if the {tag, word} is {01, 110011}, then the decoded tagless word is $\{010101 \oplus 110011\} = 100110$. This implies that the latency of the MFNW read data path is minimally affected by this simple XOR operation.

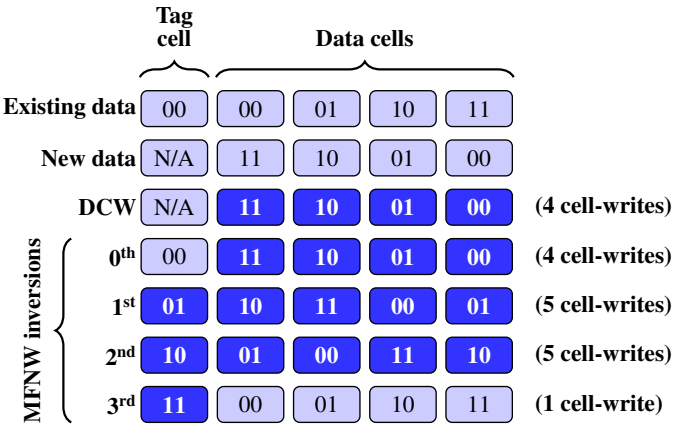

Figure 3.1: For the shown existing and new data, DCW results in 4 cell writes. CHD MFNW chooses to write the 3rd inversion of the new data, since it results in the minimum number of cell writes (1) among all other inversions. Note that cell-writes are indicated with darker-color backgrounds.

3.1.2 Cell Hamming Distance (CHD) MFNW

We define the cell Hamming distance (CHD) between two words W_1 and W_2 as the number of cells in which the two words differ from each other. For example, in MLC memory, if $W_1 = 2130_4$ and $W_2 = 2121_4$, the CHD is 2. Note that CHD(W_1, W_2) = CHD(W_2, W_1), i.e., the CHD is symmetric.

Before analyzing CHD MFNW, we provide a simple example which illustrates the potential of CHD MFNW in comparison to DCW. Fig. 3.1 shows an existing word that is to be replaced by a new word. Note that each cell is bounded in a square for readability purposes. Also note that DCW does not require a tag cell and it is therefore ignored, resulting in 4 cell writes (0% saving). CHD MFNW examines all possible inversions and picks the 3rd, since it results in the minimum number of cell writes (1). Note that this 1 cell write implies 4 'no-write' operations. We emphasize this because in our analysis, we count the average number of 'no-writes' and deduct it from the total number of cells per word (including the tag cell) to obtain the average number of cell writes. Therefore, in the following discussion, we use *the number of no-cell-writes*, or alternatively, *the number of no-writes* to refer to the number of cases where MFNW does not overwrite a cell during a write.

Probabilistic model for CHD MFNW: In the following, we develop a probabilistic model for CHD MFNW. The objective is to derive the expected number of cell writes. Without loss of

generality, this derivation assumes MLC NVM. Given a {tag,word}, note that the least significant bit (LSB) of the tag cell controls the inversion of all the LSBs in the remaining cells in the word. Similarly, the most significant bit (MSB) of the tag cell controls the inversion of all the MSBs in the remaining cells in the word. Therefore, we can partition {tag,word} into two strings of bits: The first (second) string is composed of the LSB (MSB) bits of the tag and other cells in the word. For example, $\{00, 000111\}$ can be partitioned into two bit strings $\{0, 001\}$ and $\{0, 011\}$. We refer to the first (second) bit string as the first (second) partition of a {tag,word}. Note that each one of these two partitions can be interpreted as a separate and independent instance of SLC FNW. Therefore, we wish to find the number of cell writes such that each of the two partitions has no more than $\lceil n/2 \rceil$ bit writes, which is consistent with the peak bit-write result for SLC FNW. Each trial of this 'compound' experiment has two possible outcomes: either 'cell-write' or 'no-cell-write', and a 'cell-write' occurs whenever a bit-write occurs in at least one partition.

Let Y be the random variable indicating the number of 'no-cell-write' outcomes. Let X_1 and X_2 be the random variables indicating the number of 'no-bit-write' events in the first and the second partitions of the word, respectively. Let $m = \lceil n/2 \rceil$. Then, the expected number of 'no-cell-write' events, denoted by $\overline{S}$, is

$$\overline{S} = \mathrm{E}\left[Y \mid (X_1 > m) \cap (X_2 > m) \cap (Y \geq l)\right]$$

Using probability theory, this expands to

$$\overline{S} = \sum_{k=0}^{n+1} k \frac{\mathrm{P}((Y = k) \cap (X_1 > m) \cap (X_2 > m) \cap (Y \geq l))}{\mathrm{P}((X_1 > m) \cap (X_2 > m) \cap (Y \geq l))} \tag{3.1}$$

Note that Y is constrained with a lower bound l. Setting $l = 0$ deactivates this lower bound. Observation 3.4.1 in section 3.4 proves that this lower bound equals $1 + \lfloor n/4 \rfloor$ for the MLC case.

To determine the probabilities in Eq. 3.1, consider the number of ways in which the event $((Y = k) \cap (X_1 = c_1) \cap (X_2 = c_2) \cap (Y \geq l))$ occurs, for some constants k, c_1, c_2, and l. Imagine the two partitions of the word are of length $n+1$ (including the tag bits). We perform SLC FNW on both partition, and observe the parallel outcomes of this exercise, i.e., four possibilities { NN, NW, WN, WW }, where N denotes a no-bit-write event and W denotes a bit-write event. A cell write event occurs at position i if at least one of the partitions results in W event at the same position. A no-cell-write occurs at cell position i when both partitions result in event N at bit position i.

Clearly, the sample space can be partitioned into two events: no-cell-write (NN) and cell write (not NN) events. Therefore, the number of ways in which $Y = k$ is simply $\binom{n+1}{k}$, as long as $k \geq l$, and 0 otherwise. Assuming that $c_1, c_2 > k$, the number of ways in which $X_1 = c_1$ is $\binom{n+1-k}{c_1-k}$, and this corresponds to the cases where we have NW outcomes only, since NN events were already counted in $\binom{n+1}{k}$.

The number of ways in which $X_2 = c_2$ corresponds to the number of WN outcomes only, since the NN were already counted in $\binom{n+1}{k}$, i.e., $\binom{n+1-k-(c_1-k)}{c_2-k}$ or $\binom{n+1-c_1}{c_2-k}$. Therefore, the number of ways in which $((Y = k) \cap (X_1 = c_1) \cap (X_2 = c_2) \cap (Y \geq l))$ event occurs is

$$\begin{cases} 0 & k < l \\ \binom{n+1}{k}\binom{n+1-k}{c_1-k}\binom{n+1-c_1}{c_2-k} & \text{otherwise} \end{cases}$$

Dividing the above by 4^{n+1}, which is the size of the sample space, gives us the probability of such event. Therefore, the probability of the event $((Y = k) \cap (X_1 > m) \cap (X_2 > m) \cap (Y \geq l))$, which shows up in the numerator of Eq. 3.1, is expanded as the summation of the probabilities as follows:

$$\begin{cases} 0 & k < l \\ \frac{1}{4^{n+1}} \sum_{\forall (c_1,c_2)>m} \binom{n+1}{k}\binom{n+1-k}{c_1-k}\binom{n+1-c_1}{c_2-k} & \text{otherwise} \end{cases}$$

and the denominator of Eq. 3.1 is

$$\frac{1}{4^{n+1}} \sum_{c_3 \geq l, (c_1,c_2)>m} \binom{n+1}{c_3}\binom{n+1-c_3}{c_1-c_3}\binom{n+1-c_1}{c_2-c_3}$$

Substituting these quantities into Eq. 3.1, we obtain the expected number of 'no-cell-write' events:

$$\overline{S} = \sum_{k=l}^{n+1} \frac{k \sum_{\forall (c_1,c_2)>m} \binom{n+1}{k}\binom{n+1-k}{c_1-k}\binom{n+1-c_1}{c_2-k}}{\sum_{c_3 \geq l, (c_1,c_2)>m} \binom{n+1}{c_3}\binom{n+1-c_3}{c_1-c_3}\binom{n+1-c_1}{c_2-c_3}}$$

Note that the denominator is completely independent of k, and therefore may be placed outside the summation over k. Also, the expected number of 'cell-writes', $\overline{W}$, is

$$\overline{W} = n + 1 - \overline{S} \tag{3.2}$$

To evaluate the accuracy of this approximation, we computed the average number of cell writes numerically using Monte Carlo simulation for some values of n and compared the result with

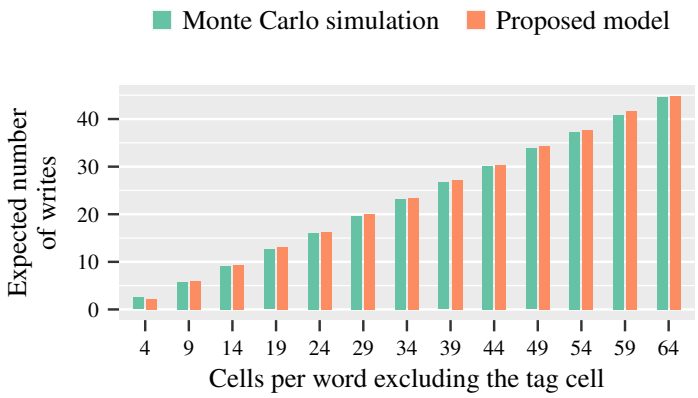

Figure 3.2: The expected number of cell writes using Monte Carlo simulation in comparison to the proposed model (Eq. 3.2).

Eq. 3.2 in Fig. 3.2. The geometric mean error is less than 1.5% and the error itself can be as low as 0.4%.

Finally, the expected write energy of CHD MFNW is $\overline{W} \times \overline{e}$, where $\overline{e}$ is the average energy over all cell states. This is true by symmetry, since the number of cell writes does not influence which cell states are written, but assumes all cell states have equal probability of occurrence. The average energy saving as a result of using CHD MFNW normalized to DCW is $1 - \frac{4\overline{W}}{3n}$.

3.1.3 Energy Hamming Distance (EHD) MFNW

We define the energy Hamming distance (EHD) between two words W_1 and W_2 as the energy required to write the cells of W_2 that are different from W_1. For example, for MLC NVMs, if $W_1 = 2130_4$ and $W_2 = 2121_4$, the EHD is the energy required to write state 2 plus the energy required to write state 1. Note that EHD$(W_1, W_2) \neq$ EHD(W_2, W_1) except when $W_1 = W_2$, unlike CHD MFNW. Clearly, write energies are technology-dependent, and therefore, without loss of generality, we base our discussion on the write energies of the MLC PCM prototype design proposed in [Bedeschi *et al.* 2009]. For this prototype, the energy in our example is 547+307=854 pJ. Table 3.1 lists cell states against their average energies as reported in the same reference.

Note that it is not necessarily true that reducing the number of cell writes leads to a reduction in write energy. Fig. 3.3 shows a real memory write scenario extracted from memory traces of

Table 3.1: Average write energies for the four MLC states of the PCM prototype [Bedeschi *et al.* 2009]

MLC state	Energy (pJ)
00	36
01	307
10	547
11	20

the 'perlbench' SPEC CPU2006 benchmark [SPEC CPU 2006]. In this example, we observe that the minimum number of cell writes, i.e., 5 writes, occurs at the 2^{nd} inversion. This is the preferred choice of CHD MFNW. On the other hand, we see that this choice leads to the highest write energy among all other inversions. The least write energy occurs if we choose 3^{rd} inversion, which results in 9 cell writes. This example clearly illustrates and motivates the necessity of EHD MFNW, which essentially chooses the inversion that results in the minimum cell write energy, regardless of the number of cell writes.

For this reason, we implement EHD MFNW, since our objective is to lower the write energy. Although this may increase the expected number of cell writes per word, it reduces the probability of programming a high energy state into a specific cell. In other words, it makes low energy states more likely to occur than high energy states. This implies that the programming effort is lowered in general, and therefore the expected lifetime of memory cells is also increased.

Unlike CHD MFNW, the analysis of the EHD MFNW is more involved. In addition to knowing if a cell write occurred, we also need to know which state has been written. In other words, we need to know the expected number of times each state was written. However, we can simplify this analysis by deriving an upper bound on the expected write energy per word. This upper bound is given by $\overline{W} \times \bar{e}$, where $\overline{W}$ is defined in Eq. 3.2 and $\bar{e}$ is the average energy over all cell states. Our results that the empirical averages of EHD MFNW write energy can be 16% lower than this bound.

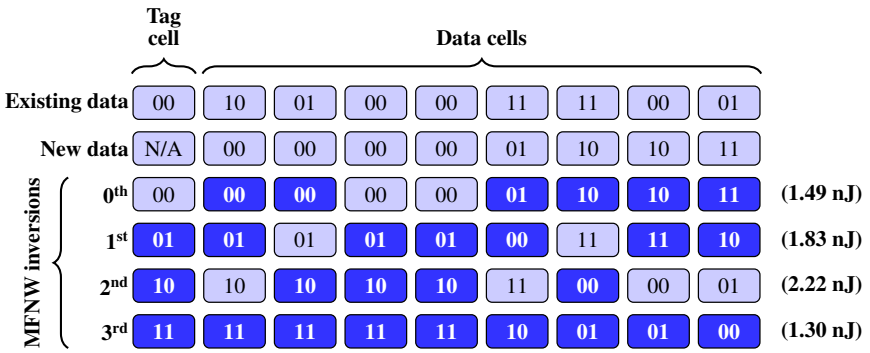

Figure 3.3: For the new and existing data above, CHD MFNW picks the 2nd inversion, resulting in minimum cell Hamming distance (but the highest energy too). EHD MFNW picks the 3rd inversion, since it results in minimum energy, regardless of cell distance.

3.1.4 TFNW: TLC Flip-N-Write

MFNW can be extended to TLC NVMs, referred to as TFNW in this work. Since a TLC can have eight states, eight inversions are generated during a memory write, and the one consuming the least energy will be selected to replace the old NVM content. We adopt an EHD-based TFNW, since the write energy consumption of the CHD version is also bounded by the EHD TFNW, as in the MLC case. Henceforth, we refer to EHD TFNW as TFNW. Evaluation of TFNW is based on the 8-state TLC RRAM prototype in [Xu *et al.* 2013]. The state-energy pairs of this prototype are shown in Table 3.2.

Table 3.2: Average write energies for the eight TLC states of the RRAM prototype [Xu *et al.* 2013]

TLC state	Energy (pJ)
000	2
001	6.7
010	19.3
011	35.1
100	35.6
101	19.6
110	8.5
111	1.5

To illustrate TFNW, consider an example with $n = 2$, i.e., the number of cells per word, excluding the tag cell, is 2. Also, assume the old content is 023_8. Note that we use base 8 (octal) to represent the words and inversions in this example. Also, in the old content, the leftmost digit is the tag (0). Consider the case where the old content as to be replaced with the new content 13_8. The first step is to calculate all the inversions of the new content:

- The 0^{th} inversion is 013_8,
- 1^{st} inversion is 102_8,
- 2^{nd} inversion is 231_8,
- 3^{rd} inversion is 320_8,
- 4^{th} inversion is 457_8,
- 5^{th} inversion is 546_8,
- 6^{th} inversion is 675_8, and
- 7^{th} inversion is 764_8.

The energies of the inversions as ordered above are: 6.7, 28, 61.1, 37.1, 56.7, 63.7, 29.6, and 45.6 pJ, respectively. Clearly, the 0^{th} inversion is selected for replacing the old content, since it results in the lowest write energy.

3.1.5 Further Energy Reductions

One way to achieve more energy reduction is to reduce n, i.e., the number of cells per word, excluding the tag cell. In this manner, the maximum energy reduction is achieved when $n = 2$, where MFNW completely avoids writing the highest energy MLC state 10 and TFNW avoids writing the highest energy TLC states 011 and 100 (for a proof of this statement, refer to observations 3.4.2 and 3.4.3 in Section 3.4). This implies an average of 47% and 40% write energy reductions in the MLC and TLC cases, respectively, due to high energy state avoidance. However, this is at the cost of 50% NVM overhead.

We can achieve higher energy reductions without this high NVM overhead. Recall that MFNW generates four inversions and selects the inversion with the minimal write energy for writing into NVM. We can achieve further write energy reductions by increasing the number of generated inversions, provided that extra NVM and controller overheads are tolerable. One can easily see

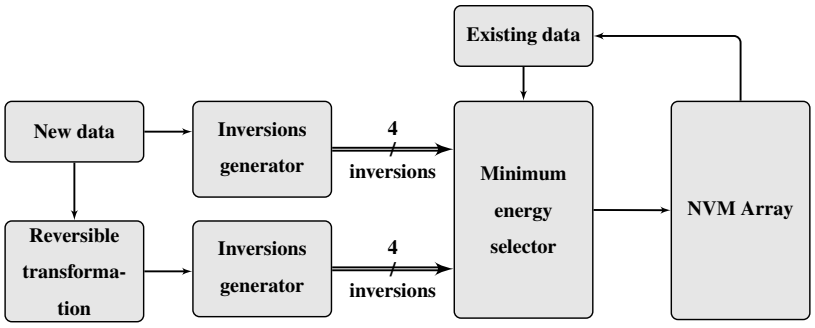

Figure 3.4: Reusing the components of MFNW to construct MFNW2.

that the maximum number of cell-level mappings is 24 in the MLC case. One approach to generate a subset of the 24 possible mappings is to utilize a lookup-table. Alternately, it is possible to reuse the components of MFNW architecture to construct more inversions as follows. Before the inversions are generated, reversible transformation(s) of the new word can be used to generate the four inversions of the original word and another four inversions of the transformed version(s) of the new word. This results in a total of $4(1+T)$ inversions (mappings), where T is the number of reversible inversions. This widens the search and increases the chances of finding a lower energy inversion. But it also implies the usage of two tag cells, one to track the selected inversions (MFNW tag cell), and another to track the selected transformation. A practical value of T is ≤ 3, and $T = 3$ results in 16 mappings. In this work, we evaluated $T = 1$ and $T = 3$, since they both result in mappings that are a power of 2. We refer to MFNW with $T = 1$ as MFNW2, and with $T = 3$ as MFNW3. Fig. 3.4 shows how MFNW2 can be constructed by reusing the components of MFNW: two inversions generators and a minimum energy selector. The internals of the inversions generator and minimum energy selector blocks are described in Section 3.2 along with the discussion of the hardware and delay overhead of MFNW.

Although many reversible mappings are possible, the primary factor in determining the expected energy reduction is the number of evaluated inversions. The choice of a certain mapping over another has only little influence on the resultant energy reduction. To demonstrate this case, we show the result of a Monte Carlo (MC) simulation for $T = 1$ and $T = 3$ and observe the change in energy reduction as a result of changing the mapping. Although many reversible map-

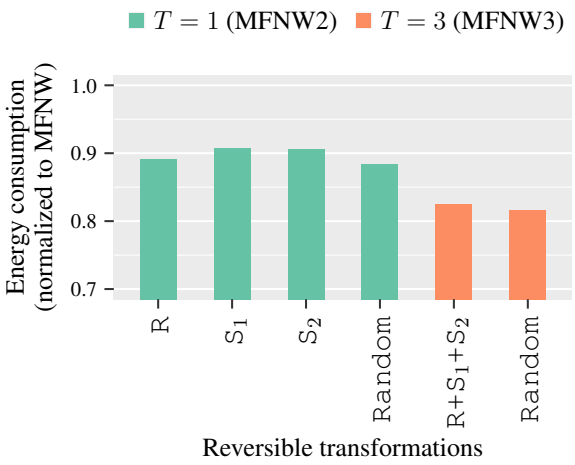

Figure 3.5: The effect of choosing a transform over the other on energy consumption. Here, R, S_1, and S_2 denote bitwise rotation, swapping MLC states 10 and 11, and swapping MLC states 01 and 11, respectively. All the bars are normalized to MFNW energy consumption.

pings were considered, we only show the result of three for brevity, i.e., bitwise rotation, swapping MLC states 10 and 11, and swapping MLC states 01 and 11 denote by R, S_1, and S_2, respectively. R rotates the new word (to be written) to the right one logical bit position. S_1 and S_2 swaps the occurrence of two MLC states. These transformations aim at changing the statistics of the states before applying the inversions. Fig. 3.5 presents MC simulation results. For comparison, we normalized energy reductions to MFNW without transformations (the first bar). Also, independent of the choice of the transform, the average energy reduction associated with it is bounded by the case where the new word and its transformation(s) are completely independent and uniformly distributed [Seyedzadeh *et al.* 2015] (the fifth and seventh bars). Clearly, the variation in energy reduction due to the choice of the transformation in MFNW2 ($T = 1$) is negligible ($< 2\%$). The sixth bar in the figure corresponds to MFNW3, in which all the three transformations are used, ending up with 16 inversions to choose from. Independent of the combinations of transforms used for MFNW3, the energy reduction is always bounded by the assumption that all the four versions of the the new word were independent and uniformly distributed (the last bar in the figure). The combinations of transforms we use for MFNW3 are only 1% more than the lower bound. This result assumes a word size of 16 cells ($n = 16$).

Although it is possible to derive similar variations of TFNW as MFNW2 and MFNW3, TFNW is already capable of achieving comparable energy reductions in comparison to state-of-the-art TLC encodings, especially when the same NVM overhead is used, as seen the results section (refer to Fig. 3.8 in Section 3.2). Therefore, we believe there is no need to use this method of energy reduction for TFNW, since it will also result in more area for the controller than MFNW2.

3.1.6 Endurance Evaluation

Memory lifetime evaluation is a challenging task with the primary challenge being the -practically- infinite time to simulate all possible wear patterns and failure scenarios. In the literature, people make a number of simplifying assumptions in order to make such a simulation possible [Schechter *et al.* 2010]. Examples include the assumption of a fixed bit flip probability and the existence of a perfect wear leveling scheme. With such simplifications, lifetime simulation can be performed in a much shorter time, with the simulator being able to track the lifetime of about 4K pages until they are completely dead. These simplifications are reasonable for the work done in [Schechter *et al.* 2010] because they are evaluating different error correction codes.

However, in this work, we are comparing the effects of different encoding solutions on NVM endurance. Modeling the encoding solutions by fixed MLC/TLC state probabilities could potentially mask the ability of these solutions in avoiding high energy states. Therefore, we avoid modeling the encoding solutions by fixed MLC/TLC state probabilities and equip our simulator with a full-fledged encoder for every encoding we evaluate in this work. Furthermore, since the assumption of a perfect wear leveling does not contribute to the evaluation of the encoding solutions, we also do not assume any wear leveling solution.

This, however, implies that our endurance simulator will take a very long time to simulate the same amount of memory as in [Schechter *et al.* 2010]. In order to work around this, we propose a word level Monte Carlo endurance simulator, in which the simulator re-writes to the same word until one of its MLCs/TLCs fails.

First, we have to assume the word length n, which is the number of cells in the word, excluding auxiliary cells, e.g., the tag cells. Next, the simulator generates a uniformly distributed random number that fits in the n-cell word. The n-cell word is next encoded and written back to the same

memory word. During the writing, the simulator keeps track of the "ages" of each cell in the memory word, including auxiliary cells. A cell's age starts with 0 and is incremented with each write to the cell. Furthermore, to make the simulator cost-aware, when a certain MLC/TLC state is written to a cell, the cell's age is incremented by an amount that is proportional to the mean energy required to write that state. As an example in the MLC case, writing state 11 results in an age increment of 1 and writing state 00 results in an age increment of $\lceil 36/20 \rceil$, where state 11 costs 20 pJ and state 00 costs 36 pJ. This loop continues until at least one of the cells reaches its lifetime, in which we report the "average" age of the cells and stop the simulation. The lifetimes of the cells are computed initially at the beginning of the simulation. PCM and RRAM cells are known to have a normally distributed lifetimes of mean 10^8 and a coefficient of variation ranging between 0.1 and 0.3 [Xu *et al.* 2013, Schechter *et al.* 2010].

This Monte Carlo endurance simulation is carried out for each version of MLC/TLC MFNW as well as state-of-the-art MLC/TLC encoding solutions. We also examine the effects of changing the word length on endurance. Results are reported in average Million writes per cell before the first failure occurs.

3.2 EVALUATION AND RESULTS

In this section, we present the simulation results for evaluating MFNW (and its derivatives MFNW2 and MFNW3) as well as TFNW on the selected MLC PCM and TLC RRAM prototypes [Bedeschi *et al.* 2009, Xu *et al.* 2013]. It should be noted that we use EHD versions of MFNW and TFNW in all our simulations, since they produce the best results. We start by describing our simulation setup and configurations. This is followed by presenting the evaluation results of the energy reductions on the MLC and TLC cases. Next we show the endurance simulation results for MLC and TLC cases. Finally, we present an estimate measure of the hardware implementation and delay overheads of MFNW and TFNW.

Simulation setup: We evaluated MFNW and TFNW using a trace-driven memory simulator. We extracted the memory traces of a number of floating point and integer SPEC CPU2006 [SPEC CPU 2006] benchmarks using the Pin binary instrumentation tool [Luk *et al.* 2005]. A typical

Table 3.3: SPEC CPU2006 benchmarks used in this work. For each benchmark, the size of the memory trace file (in MB) is shown.

Benchmark	Trace size (MB)	Benchmark	Trace size (MB)
462.libquantum	1,709	400.perlbench	829
437.leslie3d	1,709	403.gcc	808
470.lbm	1,709	444.namd	748
401.bzip2	1,709	454.calculix	728
458.sjeng	1,709	465.tonto	666
433.milc	1,709	447.dealII	488
434.zeusmp	1,709	482.sphinx3	451
481.wrf	1,646	459.GemsFDTD	441
429.mcf	1,486	464.h264ref	311
410.bwaves	1,483	450.soplex	194
473.astar	1,416	435.gromacs	186
445.gobmk	1,334	471.omnetpp	110
436.cactusADM	1,066	483.xalancbmk	31
456.hmmr	960	453.povray	18

cache system has been assumed during the generation of the traces, i.e., 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. However, during the instrumentation, only main memory access requests were recorded to generate the traces. We then sort the generated traces by the destination address (then by the write request time). In this way, the simulator has $\mathcal{O}(1)$ space-complexity, since we only need to keep track of the old content only when the current memory write request has the same address as the previous request. Once the destination address of the write request changes, the destination address of the previous write request will not appear again in the remaining part of the trace (due to sorting). For this reason, the trace simulator only needs to keep track of a single cache line at any time. Table 3.3 lists the SPEC CPU2006 benchmarks used in this work as well as the size of the memory trace generated for each benchmark. Since some of the traces are larger, we also use normalization during the calculation of the geometric mean energy reduction.

MLC PCM simulation results: We compare the geometric mean write energies across all the benchmarks for different versions of MFNW in comparison to state-of-the-art MLC PCM encoding solutions. The solutions we compare with are the energy efficient encoding (EEE) [Wang *et al.* 2011] and the two-cells-to-three-cells (TTT) mapping in [Mirhoseini *et al.* 2015]. Fig. 3.6 shows

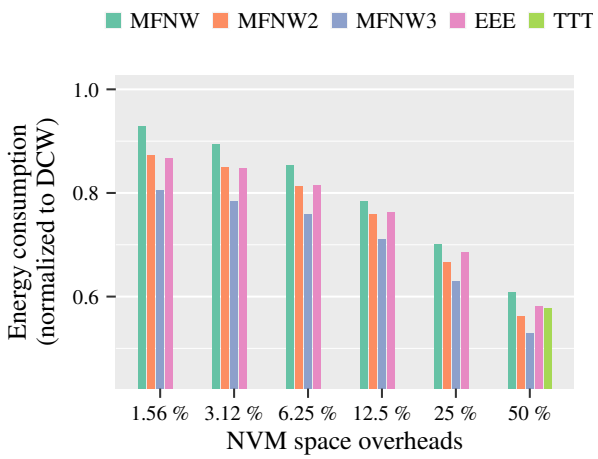

Figure 3.6: Energy consumptions of MFNW, MFNW2, MFNW3, EEE [Wang *et al.* 2011], and TTT [Mirhoseini *et al.* 2015] for different NVM overheads. Energy consumptions are all normalized to DCW energy consumption. While MFNW2 and MFNW3 provides more energy reduction than state-of-the-art encoding solutions for the same NVM overhead, MFNW in its original form can also achieve high energy reductions provided enough NVM overhead.

the simulation result. The bars in this figure are normalized to the energy consumption of MLC DCW [Yang *et al.* 2007]. Clearly, all versions of MFNW perform better as the NVM overhead increases. The energy reduction peaks at about 53% for MFNW3 for an NVM overhead of 50%. We also show in Fig. 3.7 how many times each MLC state has been written for each encoding during this simulation. We can conclude from this chart that all the encoding solutions (except TTT) perform almost the same number of writes. The major difference is the ratio at which each MLC state is written. Another observation is that the ratio at which MLC state 10 is written greatly shrinks as the NVM overhead is increased. Note that MFNW, in its original form, avoids writing the highest energy MLC state 10 altogether when enabled with an NVM overhead of 50% (Refer to observation 3.4.2 in section 3.4).

TLC RRAM simulation results: Similarly, we compare the geometric mean write energies across all the benchmarks for TFNW (TLC FNW) in comparison to incomplete data mapping (IDM) TLC encoding solution [Niu *et al.* 2013]. Fig. 3.8 shows the result. The bars in this chart are normalized to the energy consumption of TLC DCW. Clearly, TFNW performs better as the NVM overhead is increased. The energy saving peaks at about 53% for TFNW with an NVM overhead of 50%. For the same NVM overhead of IDM, TFNW's energy reduction is almost the same as IDM. Also, we

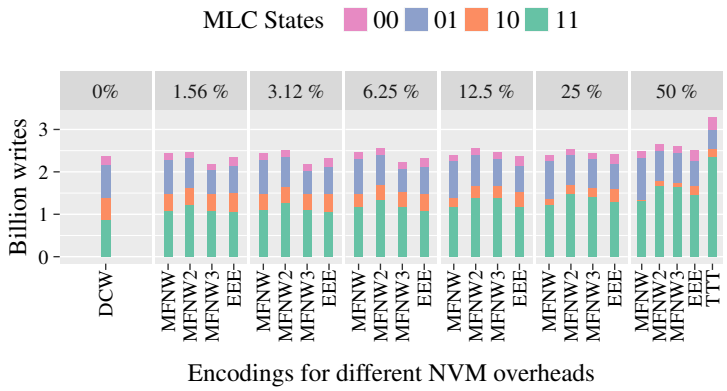

Figure 3.7: The Y-axis is the number of writes (in billions) and the X-axis lists the encoding solutions: DCW [Yang *et al.* 2007], MFNW, MFNW2, MFNW3, EEE [Wang *et al.* 2011], and TTT [Mirhoseini *et al.* 2015] for NVM overheads of 0%, 1.56%, 3.12%, 6.25%, 25%, and 50%. Each bar in this chart is subdivided into four segments of sizes corresponding to the number of times each MLC state is written.

show how many times each TLC state has been written for each encoding during this simulation (Fig. 3.9). By design, IDM avoids writing TLC states 011 and 100 altogether, however, it results in the maximum number of total cell writes. For the same NVM overhead as IDM, TFNW results in a total cell writes that is almost 30% less than IDM. Also, for this NVM overhead (20%), the highest two energy TLC states 011 and 100 are written only 10% of the time. Similar to the MLC case, when the NVM overhead is 50%, TNFW avoids writing the two highest energy TLC states 011 and 100 (See observations 3.4.2 and 3.4.3 in section 3.4).

MLC PCM endurance results: Fig. 3.10 shows the number of writes (in millions) before the first cell failure occurs for each encoding solution and for different word lengths (n). As expected, the best case for MFNW occurs at the lowest value of n. Then the number of writes gradually decreases as n increases, approaching the number of writes of MLC DCW. Note that in the cases of MLC DCW and TTT [Mirhoseini *et al.* 2015], the number of writes until the first failure is constant with respect to n. One note about this result, the reason MFNW2 has lower writes than MFNW for $n = 2$ is that we take the average cell writes for all the cells including the tag cell(s), and in MFNW2, only MLC states 00 and 11 are written to one of the tag cells, and thus, the average cell writes of the four cells is lowered.

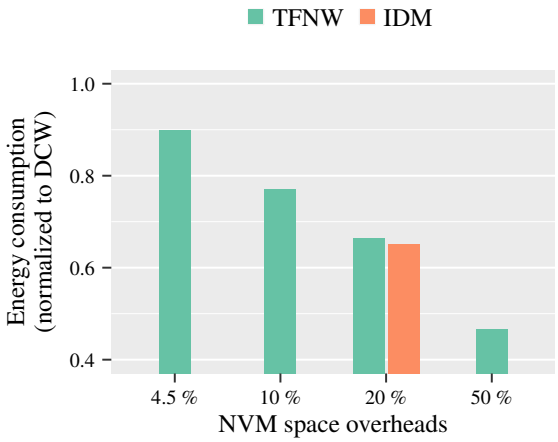

Figure 3.8: Energy consumptions of TFNW (for different NVM overheads) and IDM [Niu *et al.* 2013]. For the same NVM overhead as IDM, TFNW achieves almost the same geometric mean energy reduction. When NVM overhead is relaxed to 50%, TFNW can achieve more than 53% energy reduction. Energy consumptions are all normalized to DCW energy consumption.

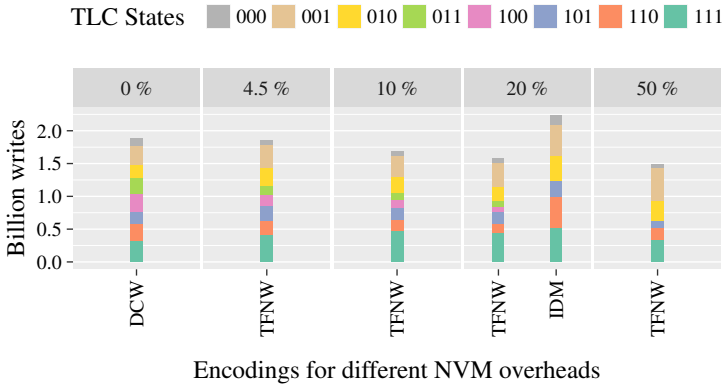

Figure 3.9: The Y-axis is the number of writes (in millions) and the X-axis lists the encoding solutions: TFNW and IDM [Niu *et al.* 2013] for different NVM overheads. Each bar in this chart is subdivided into eight segments corresponding to the number of times each TLC state is written.

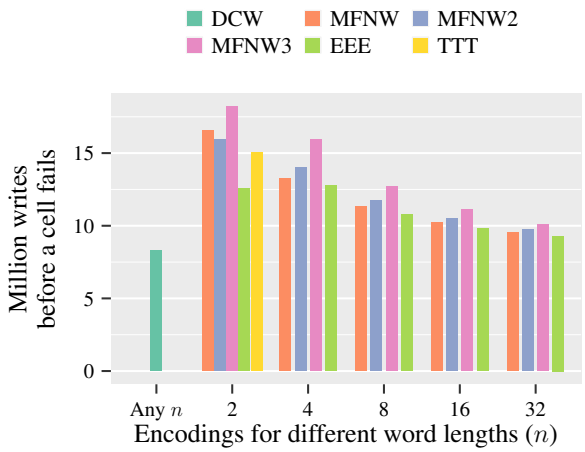

Figure 3.10: The Y-axis is the average number of (million) writes per cell before the first cell failure occurs. The X-axis lists encoding solutions: MLC DCW, MFNW, MFNW2, MFNW3, EEE [Wang *et al.* 2011], and TTT [Mirhoseini *et al.* 2015] for different word lengths. Clearly, all versions of MFNW perform better for small n. As n gets larger, the endurance effect of MFNW approaches MLC DCW's effect on endurance.

TLC RRAM endurance results: Fig. 3.11 shows the number of writes (in millions) before the first cell failure occurs for TFNW in comparison to TLC DCW and and IDM [Niu *et al.* 2013] encoding solutions and for different word lengths (n). Again, as expected, the best case for TFNW occurs at the lowest value of n. Then the number of writes gradually decreases as n increases, approaching the number of writes of TLC DCW.

Hardware and delay overheads: We implement MFNW for the MLC PCM prototype proposed in [Bedeschi *et al.* 2009]. We also choose the word length to be 8 physical cells, or 16 logical bits (excluding the tag cell), since (i) it divides 512, which is the width of our cache line, i.e., no need for padding; and (ii) it is a feasible option (with respect to space overhead) that maximizes the energy reduction in comparison to DCW. With 8 cells per word (a memory overhead of 12.5%), the lower bound on the expected energy reduction compared to DCW is 15%.

We implement Verilog modules for both read and write circuitry to support MFNW. The read module is a bitwise XOR of the tag cell replicated 8 times with the 8 cells of the word in the destination address. Only one clock cycle is required to perform MFNW read, which is also the case for DCW read, i.e., the latency overhead of MFNW read is 0 cycles, since DCW serves as the baseline for our comparisons.

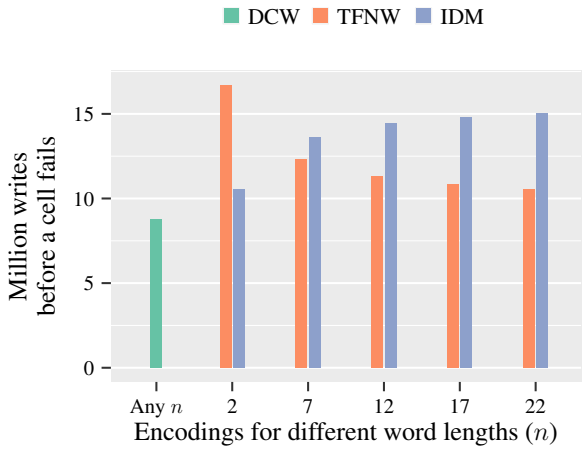

Figure 3.11: The Y-axis is the average number of (million) writes per cell before the first cell failure occurs. The X-axis lists encoding solutions: TLC DCW, TFNW, and IDM [Niu *et al.* 2013] for different word lengths. Clearly, TFNW performs better for small n. As n gets larger, the endurance effect of TFNW approaches TLC DCW's effect on endurance.

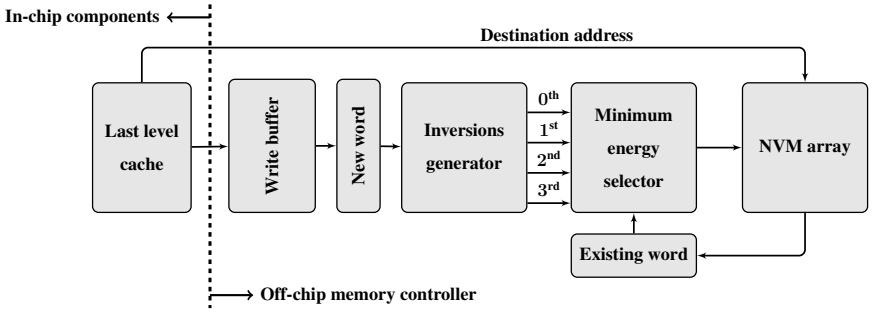

Figure 3.12: EHD MFNW write path: The cache lines are first stored in the write buffer. Every cache line in this buffer is then sliced into n-cell words, and every word is forwarded to an inversions generator block. Note that this diagram only shows one slice of the write path. The minimum energy selector chooses the inversion that results in minimum write energy taking into account the current content of the destination address. Finally the minimum energy word is forwarded to the NVM array to overwrite the word at the destination address.

The write path is shown in Fig. 3.12. When a write request arrives, the controller proceeds as follows. First, it fetches the old word, including its tag cell from memory. In parallel, it computes the four inversions of the new word and passes them to the next logic block labeled "minimum energy selector". When the inversions and the old word are both available, this block picks the inversion that results in the minimum write energy. This inversion is then forwarded to NVM array to overwrite the old data.

The logic block "minimum energy selector" starts by constructing $C(i, j)$ which counts the number of times cell state j occurs in inversion i. Note that MFNW only writes cells of the new word that are different from the old word, and therefore, we only count the cell states that result in cell writes. There are 16 such counters, 4 counters per inversion. For example, the 3rd inversion in Fig. 3.3 has the following set of counters: $C(11, 00) = 1$, $C(11, 01) = 2$, $C(11, 10) = 1$, and $C(11, 11) = 5$, where the arguments of C are shown in binary. After evaluating $C(i, j)$ for all cell states and inversions, the block evaluates the energy of each inversion i as $\sum_{j=0}^{3} C(i, j) * e(j)$, where $e(j)$ is the energy required to write cell state j. This sum of products can be simplified for this specific prototype as follows. First, we scale down the cell state energies by dividing them by 10, and rounding to the nearest integer, resulting in write energies 4, 31, 55, and 2 for cell states 00, 01, 10, and 11, respectively. Note that this does not affect the choice of the minimum, since all the energy numbers are scaled down by the same factor. Second, we round these numbers to the nearest power of 2 resulting in: 4, 32, 64, and 2. This simplifies the multiply-add operations to shift-add operations. The resulting write path after this simplification incurs only 3 clock cycles per write operation, which is two additional extra clock cycles in comparison to DCW.

The logic overhead of MFNW, which is roughly 10k gates, is negligible in comparison to the area of state-of-the-art non-volatile memories, e.g., [Choi *et al.* 2012]. MFNW2 has roughly double the logic overhead of MFNW and MFNW3 has roughly double the overhead of MFNW2. Finally, since TFNW generates eight inversions just like MFNW2, its logic overhead is approximately the same as MFNW2.

3.3 CONCLUSIONS

In this chapter, we present MLC and TLC versions of SLC FNW, referred to as MFNW and TFNW, respectively. Both can effectively reduce energy consumption and improve the endurance of MLC and TLC NVM technologies. The fundamental idea behind the efficiency of our proposed solutions is the usage of cell inversions, which replaces bit flipping in SLC FNW. We also show how to achieve further energy reductions using MFNW2 and MFNW3, i.e., extensions to the original MFNW. The MLC (TLC) results of MFNW3 (TFNW) report an energy reduction of 19% – 47% (10% – 53%).

We also present an endurance model that we use for the lifetime simulation of MLC/TLC NVMs. The model can reveal the effects of different encoding techniques for the sake of evaluating the endurance of MLC/TLC NVMs. The results indicate that MFNW3 (TFNW) can prolong the lifetime by up to 100% (87%).

The choice of the word length (and hence the NVM overhead) in MFNW and TFNW plays an important role in determining the amount of energy saving. The peak saving of MFNW and TFNW is when the NVM overhead is 50%. As we lower the NVM overhead, the energy saving is also reduced. We believe that the original MFNW can be more beneficial if NVM overhead is $\geq 12.5\%$, where it achieves an average energy saving of $\geq 21.5\%$ over DCW. However, if this overhead is not an option, one can consider using MFNW2 or MFNW3 instead as both can achieve the same energy saving as MFNW ($\geq 21.5\%$), but with much lower NVM overheads of 6.25% and 1.56%, respectively.

3.4 APPENDIX

This supplementary section provides some observations (and proofs) about MFNW/TFNW. We start by the following observation, which establishes an upper bound on the number of writes on CHD MFNW.

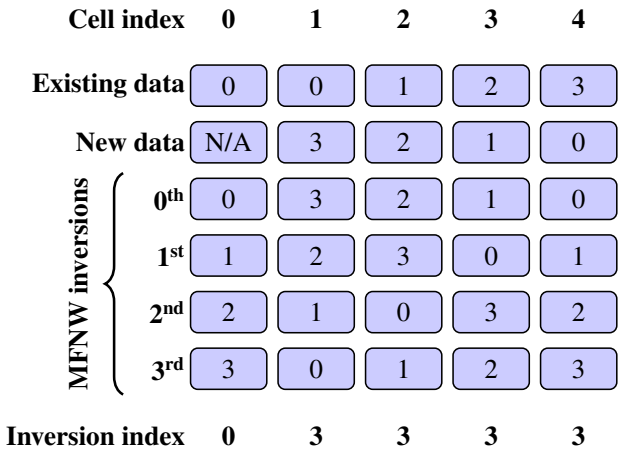

Figure 3.13: Illustration of the inversion index.

Observation 3.4.1. *Assuming n-cell per word, excluding the tag cell, and m-state per cell, the number of 'no-cell-write' events of CHD MFNW is $\geq 1 + \lfloor n/m \rfloor$. Equivalently, the number of 'cell-write' events is $\leq n - \lfloor n/m \rfloor$.*

Proof. We prove this observation by mathematical induction. However, let us first develop the notion of *inversion index* that is used in the proof.

Suppose CHD MFNW is about to overwrite an old word by a new word, assuming n-cell word length (excluding the tag cell) m-state MLC NVM. Without loss of generality, we will limit m to 4 in this discussion. Suppose also that we index the cell positions of the old word and the corresponding positions of the cells of the inversions of the new word, starting from 0 (for the tag cell), 1 for the most-significant cell, ... etc. The *inversion index* is defined with respect to each column of this matrix, where the i^{th} column is composed of the inversions of the new cell at index i, and its value equals the inversion number of that cell which results in 'no-cell-write' event.

We illustrate this notation in Fig. 3.13. The first row lists cell indices, where index 0 is assigned to tag cells. The second row lists the cells of the old word, starting from the tag cell. The next four consecutive rows list the four inversions of the new word. The last row lists the inversion indices of each column. For example, the inversion index of cell number 0 (i.e., the tag cell) is 0, since the 0^{th} inversion results in 'no-cell-write' event for this column (the tag cell column). We can see from

this example that CHD MFNW performs very well if the 3^{rd} inversion is chosen, since it results in saving four cell-writes. In general, we would like to have the inversion indices to repeat as much as possible, as this means that one of the inversions saves as many writes as the most repeated inversion index.

However, for the sake of proving the upper bound on the number of 'cell-write' events, we are more interested in constructing a worst case scenario, and showing that the number of 'cell-write' events in this case is no more than $n - \lfloor n/4 \rfloor$. Since we know that each column will have exactly one inversion index, the worst case scenario occurs when the inversion indices have the least repetition.

After this terminology introduction, we can now proceed to the proof of observation 3.4.1.

Basis step: The upper bound when $n < 4$ is trivially satisfied, since W, the number of 'cell-write' events, is no more than n. This is true because we can always avoid writing to the tag cell -at least-, resulting in at most n writes.

When $n = 4$, the worst case scenario can be constructed by making sure the inversion indices are all different. Note that is not possible because $n = 4$ means that the word is composed of 5 cells. And according to the pigeon-hole principle, there exists one inversion index (at least) that is repeated. To make sure that this is the worst case, we have to ensure that exactly one inversion index is repeated. This implies that there is one inversion in which we can save 2 writes. Since this is the worst case for $n = 4$, we conclude that $W \leq 3 = n - \lfloor n/4 \rfloor$.

We need to make sure that the scenario is still a worst case scenario when we increase n, and this is achieved by minimizing the maximum repetition among the inversion indices. This is best illustrated by an example as follows. Suppose that the inversion indices for the case $n = 4$ are $\begin{bmatrix} 0 & 2 & 3 & 0 & 1 \end{bmatrix}$. Note that inversion index 0 is repeated twice, therefore, if we would like to create a worst case for $n = 5$, we can not choose inversion index 0 again, since it results in increasing the repetition of inversion index 0, but we can choose indices 1,2, or 3. This means that the number of saved writes does not change for $n = 5$ in the worst case from the case $n = 4$, even though the number of cells increased. This results in the bound $W \leq 6 - 2 = 4 = n - \lfloor n/4 \rfloor$. We keep adding more cells and assign the new inversion indices to states such that the cases are still worst case, until we reach $n = 7$, where all inversion indices are repeated twice, in which $W \leq 8 - 2 = 6 = n - \lfloor n/4 \rfloor$. From here, no matter which inversion index we choose for the

case $n = 8$, it will always result in repeating one of the inversion indices 3 times, resulting in $W \leq 9 - 3 = 6 = n - \lfloor n/4 \rfloor$.

Induction step: Let us denote the statement we would like to prove by $T(n)$, which basically states that W_n, the number of 'cell-write' events when the word is n cells wide (excluding the tag cell), is no more than $n - \lfloor n/4 \rfloor$.

Let us assume that $T(n)$ is true for some $n \geq 4$, i.e., $W_n \leq n - \lfloor n/4 \rfloor$. Our objective is to prove that $T(n+1)$ is true given $T(n)$ is true. If we add one more cell to the word and follow the worst case construction procedure described above, the maximum repetition of the inversion index is either not changed or gets incremented by one, depending on whether $n+1$ is a multiple of 4 or not. Recall that the maximum repetition of the inversion indices represents the number of 'no-cell-write' events, therefore, if it remains the same while the number of cells has been increased, it means that the number of writes has been increased by 1, i.e., $W_{n+1} \leq n + 1 - \lfloor n/4 \rfloor$, and since $n+1$ is not a multiple of 4, we conclude that $W_{n+1} \leq n + 1 - \lfloor (n+1)/4 \rfloor$.

On the other hand, if the maximum repetition of the inversion indices gets incremented by one after adding one more cell, it means that the number of writes remains the same as W_n, i.e.,

$$\begin{aligned} W_{n+1} = W_n &\leq n - \lfloor n/4 \rfloor \\ &= n + 1 - 1 - \lfloor n/4 \rfloor \\ &= n + 1 - \lfloor n/4 + 1 \rfloor \\ &= n + 1 - \lfloor (n+1)/4 + 3/4 \rfloor \\ &= n + 1 - \lfloor (n+1)/4 \rfloor \end{aligned}$$

Here, the last equality follows from $n+1$ being a multiple of 4. Therefore, $T(n+1)$ is always true as long as $T(n)$ is true for $n \geq 4$. □

Observation 3.4.2. *Assuming a word size of two MLC cells, excluding the tag cell(s), MFNW will never write MLC state 10 and will write state 01 at most once for MLC PCM prototype in [Bedeschi et al. 2009].*

Proof. First of all, we rule out trivial cases. Note that it is impossible -in this case- to write MLC state 10 two times, as this implies that the minimum energy inversion was not chosen. In other words, writing MLC state 10 two times implies that we have not chosen the minimum energy inversion, since, for instance, the second inversion of the same word results in writing MLC state 10 just once. Similarly, we can also rule out the case where MLC state 01 has been written two times, as this will also violate the minimum inversion selection. Therefore, we can safely assume that MLC state 10 has been written exactly one time and reach to a contradiction as follows. Assuming that MLC state 10 is written exactly once, the other cell can be one of the MLC states 00, 01, or 11. In case of 00 or 11, we can see that the third inversion results in at most 363 pJ, which is less than 547 pJ, i.e., the lower bound energy if MLC state 10 is written. Similarly, if the other cell was 01, then the first inversion will also result in an energy of at most 363 pJ. This implies that the assumption that MLC state 10 is written once is a false statement. Therefore, MLC state 10 will never be written in this setup of MFNW. □

Observation 3.4.3. *Assuming a word size of two TLC cells, excluding the tag cell(s), TFNW will never write TLC states 011 and 100 for TLC RRAM prototype in [Xu et al. 2013].*

Proof is similar to the proof of observation 3.4.2 and, therefore, omitted.

4.0 AN OFFLINE FREQUENT VALUE ENCODING FOR ENERGY-EFFICIENT MLC/TLC NON-VOLATILE MEMORIES

Frequent value encoding (FVE) was originally introduced to encode data and address buses [Yang and Gupta 2002, Yang *et al.* 2004, Suresh *et al.* 2009]. FVE can achieve high reductions in NVM write energy by mapping frequent values (words) into low energy codewords. Recently, FVE has been proposed to encode single-level cell (SLC) memory words in PCM [Sun *et al.* 2011]. The authors investigated both static (offline) and dynamic (online) encoding. They report that although offline encoding (originally introduced as "find-once for a given program" in [Yang and Gupta 2002]) achieves higher energy reductions on average, it requires compiler and operating system support, which is not desirable in practice. In contrast, online encoding may be better, but it requires a non-trivial online profiling effort that may impact system performance. However, to the best of our knowledge, FVE has not been applied to MLC/TLC NVMs, where a single physical cell can store more than one logical bit to realize density and cost advantages [Kang *et al.* 2008, Xu *et al.* 2013]. Since MLC/TLC NVMs have higher write energy and access latency as well as lower endurance in comparison to SLC NVMs, there is strong motivation to develop data encoding and wear-leveling techniques for MLC/TLC NVMs [Wang *et al.* 2011, Niu *et al.* 2013, Wen *et al.* 2014, Alsuwaiyan and Mohanram 2015, Jiang *et al.* 2012a].

In this chapter, we present an offline FVE for MLC/TLC NVMs that achieves an average write energy reduction approaching, and sometimes exceeding, that of optimal offline encoding [Yang and Gupta 2002]. The proposed method, which does not require compiler or operating system support, is based on finding multiple optimal FVEs. Each FVE is derived by aggregating the data frequency profiles of a group of compatible applications. To the best of our knowledge, this is the first work that presents a feasible version of an offline FVE approaching the average energy reductions of optimal offline FVE. The following contributions are presented in this chapter:

- It describes the use of k-medoids algorithm [Park and Jun 2009] to cluster a set of general-purpose benchmark applications (SPEC CPU2006 [SPEC CPU 2006])into k compatible groups. The applications are represented as an observation matrix consisting of rows of feature sets that are essential inputs to the k-medoids algorithm (Section 4.1.1).

- We process the output of the k-medoids algorithm to produce offline, low overhead, energy-efficient FVEs. The objective is to reach the write energy of optimal offline FVE while avoiding its disadvantages (Section 4.1.2). The proposed codec (coder/decoder) architecture uses read-only memories (ROMs) that are known to consume low power/energy in comparison to lookup tables (LUTs) and content-addressable memories (CAMs) (Section 4.1.3).

We divide the set of SPEC CPU2006 benchmarks into training and evaluation sets. The training set is used to derive the k offline FVE mappings, which are then used to encode the applications in the evaluation set. We evaluate the proposed solution for the MLC PCM prototype in [Bedeschi *et al.* 2009] and the TLC RRAM prototype in [Xu *et al.* 2013]. Results (Section 4.2) indicate that average write energy in the MLC case is only 5% more than that of optimal offline FVE. The result is even better in the TLC case, where the average write energy of the proposed technique is 1% less than that of optimal offline FVE. In comparison to the write energies of MLC and TLC data comparison write (DCW) [Yang *et al.* 2007], the proposed solution achieves 39% and 35% energy savings, respectively, while the memory overhead in both cases does not exceed 3.5%. We report results for different values of k (the number of clusters), and for some values, our method results in average write energy that is only 4% more than that of the state-of-the-art MLC PCM encoding technique in [Mirhoseini *et al.* 2015]. However, [Mirhoseini *et al.* 2015] requires 50% NVM overhead, which is 16$\times$ in comparison to the proposed solution. Whereas the 50% overhead of [Mirhoseini *et al.* 2015] can be reduced, we argue that such a reduction will result in exponentially larger sizes of LUTs in comparison to the codec ROMs required by our solution. For TLC RRAM, our method results in the same average write energy as incomplete data mapping (IDM) [Niu *et al.* 2013]. However, IDM has a memory overhead of 20%, i.e., 5.7$\times$ the overhead of the solution proposed in this chapter.

4.1 CONTRIBUTIONS

The main contribution presented in this chapter is the integration of an offline FVE solution for the purpose of MLC/TLC NVM write energy reduction approaching the reductions of perfect knowledge FVE without its disadvantages: the need for operating system and compiler support. Specifically, this work makes the following contributions:

- We use the k-medoids algorithm to cluster a set of applications into k compatible subsets. Precisely, our contribution is encoding the input applications as an observation matrix that consists of rows of feature sets, which is an essential input to the k-medoids algorithm (Section 4.1.1).

- We process the output of the k-medoids algorithm to derive a set of offline frequent value encodings. The objective is to achieve an average write energy that is as close as possible to the average write energy of perfect knowledge FVE (Section 4.1.2). We propose a codec architecture that utilizes ROMs that are known to consume low dynamic power in comparison to LUTs and CAMs (Section 4.1.3).

The proposed solution requires a slight modification to the traditional NVM memory word structure. This modified word structure is only internal to our codec and does not require any architectural changes outside the boundaries of the codec. As in Fig. 4.1a, an NVM word is composed of one or two tag cells, followed by a number of fixed length data slices. Each slice stores a frequent value in one of k encoded forms, and the tag cell(s) helps in the decoding process. The logical width of a data slice is referred to as frequent value length (FVL), and is equal to $\lceil log_2 v_{\max} \rceil$, where $v_{\max}$ is the maximum possible frequent value. The FVL parameter and the number of possible frequent values to be encoded (FVN) characterize FVE. In this work, we fix FVL to 8 logical bits in the MLC case (Fig. 4.1b) and 9 logical bits in the TLC case (Fig. 4.1c). For FVN, however, the obvious choice would be 2^{FVL}, and this is used in this work. It is possible, however, that FVN $< 2^{\text{FVL}}$, and this can be combined with compression to lower the number of bit writes, e.g., [Sun *et al.* 2011]. However, the cells in which the compressed values are written will potentially wear out faster. Hence, in this work, FVN is fixed to 2^{FVL}, i.e., 256 for MLC and 512 for TLC, to avoid biased wear of the memory cells.

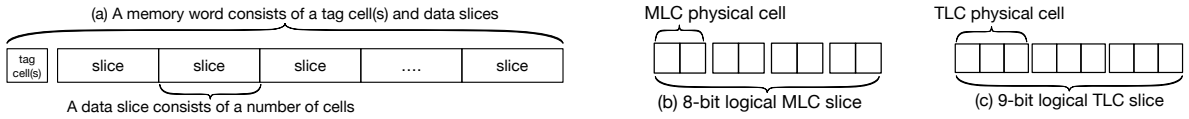

Figure 4.1: (a) A memory word in the proposed scheme consists of one or two tag cells and a number of data slices, which varies for MLC and TLC as shown in Table 4.1. Throughout this chapter: (b) an MLC data slice consists of 4 physical MLC cells or 8 logical bits and (c) a TLC data slice consists of 3 physical TLC cells or 9 logical bits.

4.1.1 Memory Trace Clustering

Finding data clusters is an unsupervised learning process in which data is divided into a given number of subgroups. Classical clustering algorithms include k-means [Wu 2012] and k-medoids [Park and Jun 2009]. Clustering has played an important role in many areas, including artificial intelligence, pattern recognition, medical research, business intelligence, psychology, and political science [Kaufman and Rousseeuw 1990, Wu 2012].

We utilize the k-medoids algorithm to cluster 32 memory traces of SPEC CPU2006 [SPEC CPU 2006] and Splash-2 [Arnold *et al.* 1992] benchmarks. These benchmarks represent a wide range of real user applications, and it is expected that most applications will have similar workloads. The k-medoids algorithm takes the following inputs: the number of clusters k, an observation matrix, and a similarity distance metric. For this work, the observation matrix consists of 32 rows (one row per benchmark), while the number of columns equals the feature set size, which is 256 for MLC and 512 for TLC.

Feature set extraction: The first step in generating the feature set of a given application is to extract its frequency information. Let $f(v)$ be a function returning the frequency of value v in the given application, where $0 \leq v <$ FVN. Using this function, we form a set of pairs $\{(v, f(v)) : 0 \leq v <$ FVN$\}$ and sort it in decreasing order with respect to $f(v)$ to generate an ordered list of these pairs. Each pair in the ordered list is composed of two entries: a value v and its frequency $f(v)$. Augmenting all the first entries of the pairs in the ordered list results in the feature set row vector $\vec{v_d}$. This feature set vector uniquely characterizes the given application. Two applications have the same vector *iff* they share similar, if not identical, frequency profiles. The rows of the

observation matrix are constructed from these feature set vectors for all the 32 benchmarks. Note that a feature set vector is a permutation of all the values from 0 to FVN-1.

The similarity metric provides a means of measuring the distance between two applications, or more precisely, two feature sets or observations. It helps assign an observation to a cluster. In this work, since the rows of the observation matrix are permutations of the same set, a rank-based metric (Spearman rank correlation) was adopted. However, in most cases (see Section 4.2), other classic metrics, which are not rank correlated, perform equally well, i.e., the cosine and square Euclidean metrics.

As illustrated in Fig. 4.2, given k, an observation matrix, and a similarity metric, the k-medoids algorithm assigns a cluster ID (ranging from 0 to $k-1$) to every observation. Obviously, the size of the clusters may not be equal. Also, some of the clusters may have a single observation. For the sake of separating training data from evaluation data, we split the output of the k-medoids algorithm into a training set and an evaluation set. This strategy is widely used in the literature and its purpose is to measure and compare the quality of clustering in both sets. A good clustering results in relatively close qualities, provided that the right number of clusters is chosen.

Since the size of the clusters may be different, and some clusters may contain an odd number of observations, we cannot always choose half the observations in each cluster to construct the training set. Instead, we create k lists, $L[i]$, where each list is initially identical to the corresponding cluster. We remove one observation at a time from the list containing the largest number of observations until the number of observations in the lists is halved. We assume that L is available as input to the code generation algorithm.The observations in the cluster lists represent the training set and the remaining observations represent the evaluation set.

4.1.2 Code Generation

Consider a function $e(v)$ that calculates the energy of frequent value v as the sum of the write energies of the cells composing v. For example, with respect to the PCM MLC prototype in [Bedeschi *et al.* 2009], if $v = 0$ and FVL=8, then $e(0) = 36 \times 4 = 144$ pJ, since the value 0 is composed of 4 MLCs whose values are all 00. Another example is $e(0) = 2 \times 3 = 6$ pJ in case of RRAM TLC [Xu *et al.* 2013].

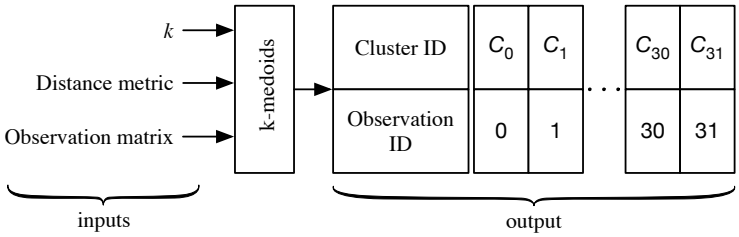

Figure 4.2: Given the number of clusters k, the required similarity metric, and the observation matrix, the k-medoids algorithm divides the 32 observations (applications) into k clusters ($0 \leq C_i < k$ and $0 \leq i < 32$, where C_i is the cluster ID of observation i.)

Next, we form a set of pairs $\{(v, e(v)) : 0 \leq v < \text{FVN}\}$ and sort it in increasing order by $e(v)$ to generate an ordered list of these pairs. Augmenting the first entries of all these ordered pairs, i.e., the v portion of the pairs, results in the vector $\vec{v_e}$. Similar to $\vec{v_d}$, the elements of $\vec{v_e}$ form a permutation of the integers in the range 0 to FVN-1. Note that $\vec{v_e}$ depends on the underlying MLC/TLC NVM technology and not on the application. It is a list of all the codewords ordered in increasing order by their energies. In the code generation algorithm, this vector is referenced item-wise, i.e., $\vec{v_e}[i]$ refers to the i^{th} element in the vector. Also, in the same algorithm, we assume that the function $F_p(o, v)$ returns the perfect knowledge frequency of value v in observation (application) o. Further, $L[i]$ is the i^{th} cluster list introduced earlier.

Algorithm 2 outlines the process of code generation. The final output of the algorithm is a two-dimensional mapping $M(i, v)$, where $0 \leq i < k$ and $0 \leq v <$ FVN are the encoding number and the value-to-be-encoded, respectively. Each cluster list generates one code as follows. First, the perfect knowledge frequency profiles of the applications in the cluster are aggregated, resulting in a set of pairs $\{(v, f(v)) : \forall v\}$. This set is sorted in decreasing order by frequency. The sorted list is stored in structure P, where $P[j]$ is the j^{th} pair with respect to the sort order, and $P[j].v$ refers to the v entry of the pair. The final step to build the i^{th} dictionary, $M(i, v)$, for all v. This is achieved by associating $P[j].v$ with the j^{th} codeword, $\vec{v_e}[j]$. These steps are repeated to build the codes for each of the other cluster lists.

Algorithm 2: Generating the k codes for the k clusters

Output : $M(i, v), 0 \leq i < k, 0 \leq v <$ FVN, the i^{th} codeword of value v

for $i = 0$ *to* $k - 1$ **do**
  Initialize $f(v) = 0, \forall v$
  for *each observation o in* $L[i]$ **do**
    for $v = 0$ *to FVN* $- 1$ **do**
      $f(v) \leftarrow f(v) + F_p(o, v)$
  Sort the pairs $\{(v, f(v)) : \forall v\}$ in descending order by $f(v)$, and let $P[i]$ be the i^{th} pair of this list. Further, let $P[i].v$ is the value part of the pair
  for $j = 0$ *to FVN-1* **do**
    $M(i, P[j].v) \leftarrow \vec{v_e}[j]$

4.1.3 Hardware Realization

The resulting codes from algorithm 2 are used to program k ROMs, one ROM per cluster. ROM i is programmed with the portion of the mapping tables corresponding to cluster i. Each slice of the incoming memory word is associated with its own encoding ROM, and the incoming memory word is encoded k times, one time per code. Then, the encoded version that results in minimum energy is selected for writing, taking into account the current content of the target address. Fig. 4.3 shows the encoding path for $k = 2$. Since we have two clusters, the code assignment algorithm will result in two mappings. Tag cell T_0 (T_1) takes the value 0 (1) to indicate that the word is encoded using the 0^{th} (1^{st}) mapping. The encoding overhead in bits due to using ROMs equals the size of one encoding ROM $\times$ the number of slices per word $\times$ the number of words in the memory line $\times$ k. Note that all encoding ROMs have the same size which is FVN $\times$ FVL bits.

For decoding, we invert the mapping tables into ROMs as shown in the decoding architecture in Fig. 4.4. To avoid extra delay in the read path, the same ROM is duplicated in each slice across the whole memory line. One or more tag cells store the cluster number to decode the encoded slices inside the NVM array. Given a tag cell and an encoded slice, a decoding ROM outputs the decoded frequent value. Despite the duplication, the overhead is low in practice. The decoding overhead equals the size of one ROM $\times$ the number of slices per word $\times$ the number of words in the memory line, and ROM size is $k \times$ FVN $\times$ FVL bits.

In Fig. 4.3 and 4.4, encoders and decoders are duplicated across all the slices in the memory line for simplicity. If this design is not affordable, an alternative lower overhead design is also possible using multiplexers and pipelining.

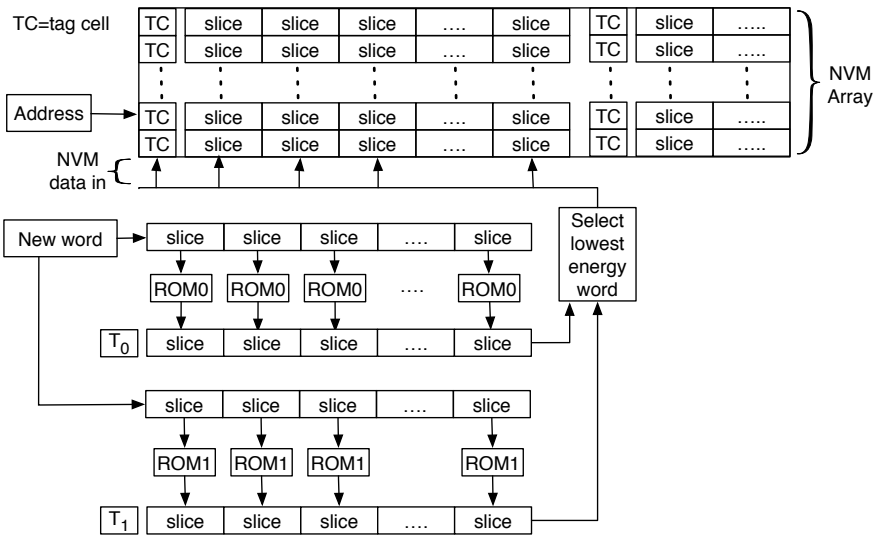

Figure 4.3: An NVM array consists of a number of rows, and each row contains words that are organized as in Fig. 4.1a. An unencoded new word from the CPU is received along with a target address. Then k encoded versions are generated and the word with the lowest energy (with respect to the current word at the target address) is selected for writing. For decoding purposes, each encoded version is associated with a tag value that is stored with the lowest energy version.

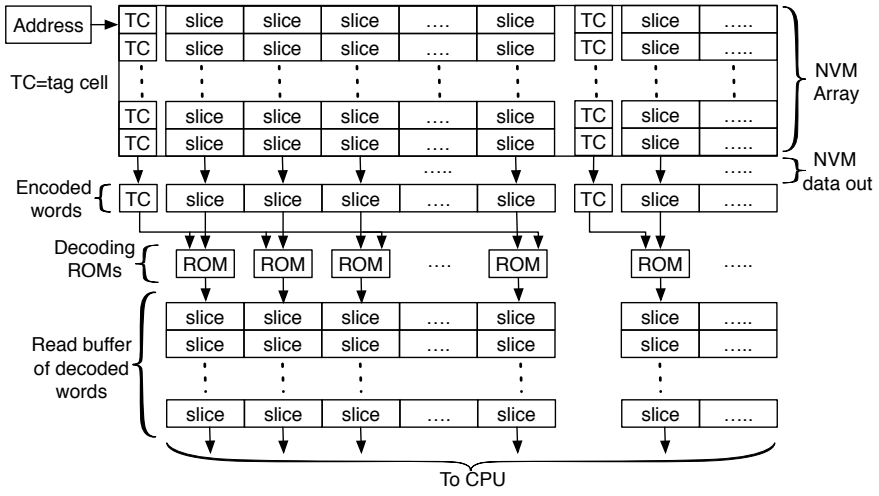

Figure 4.4: Decoding ROMs are replicated for each slice across the memory line. ROMs are addressed by slice value augmented with tag value.

Table 4.1: MLC/TLC memory line organization and NVM overhead

	MLC	TLC
Logical bits per memory line	512	513
Slice width or FVL (in logical bits)	8	9
Slices per word	8 / 16	19
Words per memory line	8 / 4	3
Tag cell(s) per word	1 / 2	1 / 2
NVM overhead	3.1%	1.8% / 3.5%

4.2 EVALUATION AND RESULTS

In this section, we present the simulation results of the proposed solution for the 4-state MLC PCM prototype in [Bedeschi *et al.* 2009] and the 8-state TLC RRAM prototype in [Xu *et al.* 2013]. For brevity, we refer to these prototypes in this section as MLC and TLC, respectively.

Simulation setup: We evaluated the proposed solution using a trace-driven memory simulator. We used the Pin binary instrumentation tool [Luk *et al.* 2005] to extract the memory traces of 32 distinct applications: 28 SPEC CPU2006 [SPEC CPU 2006] benchmarks and 4 Splash-2 [Arnold *et al.* 1992] benchmarks. Table 4.1 lists the simulation configuration and the memory overhead due to using the tag cell(s). The number of logical bits in the memory line is 512 bits for MLC and 513 bits in TLC due to padding. In MLC, a 64-bit logical word is constructed from 32 physical cells. To avoid unnecessary padding in the TLC case, we chose a word size of 57 physical cells (171 logical bits). Note that this word size is local to the memory controller and does not imply a change to the word size of the CPU.

Memory overhead: As indicated in Table 4.1, NVM overhead due to tag cell(s) is always 3.1% in MLC, since the ratio between the number of tag and data cells in the word is fixed. In TLC, the number of slices per word is constant, but the number of tag cells is one or two, and therefore, the overhead is 1.8% or 3.5%, respectively. The number of tag cells in the MLC and TLC cases is $\lceil log_2 k \rceil$, where k is the number of clusters. Further, Table 4.2 reports the overhead for the codec ROMs for different values of k. Clearly, the codec overhead is negligible in comparison to the sizes of state-of-the-art NVMs.

Table 4.2: Assuming one codec is shared among all the words in a memory line, we show the ROM overhead for MLC/TLC NVMs for different values of k.

k	2	4	8	16
MLC (KBytes)	8	16	64	128
TLC (KBytes)	43	86	171	342

Hardware and latency overheads: We implemented Verilog modules for our codec circuitry. In our design, we avoid replicating the codec across all the words of the memory line, since this results in higher area. Instead, a single codec instance is shared (and pipelined to avoid latency penalty) among all the memory words across the memory line. Using Design Compiler, we synthesize these modules on a 45nm technology node [Stine *et al.* 2007]. Since $k = 8$ provides best results for MLC, we chose to synthesize this case to examine the overheads. We implement the codec ROMs using `case` statements and the the cosine metric.

The area of the synthesized codec is 0.23 mm^2, which is negligible in comparison to the area of the state-of-the-art PCM memories, e.g., less than 0.4% the area of the 8Gb PCM in [Choi *et al.* 2012]. The latencies (energies) of the decoder and encoder paths are 1.93 ns (1.1 pJ/cell) and 3.91 ns (2.3 pJ/cell), which is also negligible in comparison to the PCM program-and-verify (P&V) write latencies (energies) [Bedeschi *et al.* 2009]. Assuming P&V writes 34 cells at once, we can reduce the total memory line latency to a single word latency, since by the time the P&V modules complete writing one PCM word, the encoder will have encoded the entire line. But the decoding latency of the complete memory line equals the number of words per line times the word latency, i.e., 15.44 ns. If this is high, the decoder can be replicated to achieve a read latency that is as low as 1.93 ns for the memory line. Although this replication results in tripling the codec area (0.62 mm^2), it also reduces the read latency by 8×.

Energy reductions: Fig. 4.5 compares state-of-the-art methods to the proposed solution for MLC PCM. The bars represent the geometric mean (GM) of the write energy for each method across all the 32 applications. GMs are normalized to the GM energy consumed by DCW [Yang *et al.* 2007]. The methods from left to right are: DCW, the 50% overhead method [Mirhoseini *et al.* 2015], MFNW [Alsuwaiyan and Mohanram 2015], cell remapping [Wang *et al.* 2011], and perfect

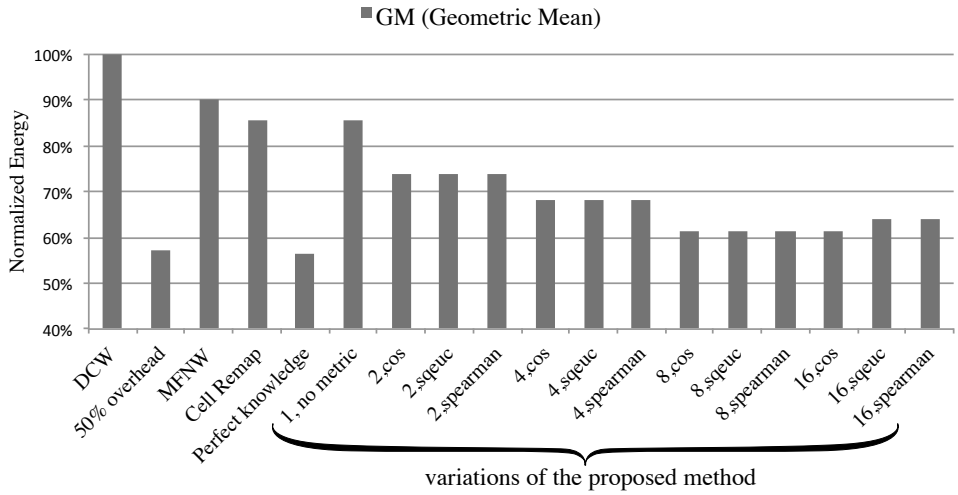

Figure 4.5: The x-axis lists state-of-the-art MLC PCM encoding techniques followed by our proposed method. The y-axis is the geometric mean (GM) write energy normalized to data comparison write (DCW) [Yang *et al.* 2007]. From left to right, the encodings are: DCW, 50% overhead encoding [Mirhoseini *et al.* 2015], MFNW [Alsuwaiyan and Mohanram 2015], cell remapping [Wang *et al.* 2011], and perfect knowledge FVE [Yang and Gupta 2002]. This is followed by aggregate FVE without clustering (1,no metric), and variations of our method, broken down by the number of clusters and distance metric, e.g., (2,cos) to indicate two clusters with the cosine metric.

knowledge FVE [Yang and Gupta 2002]. NVM overheads of these methods are 0%, 50%, 3.1%, 3.1%, and 0%, respectively. The following bars belong to the proposed solution, labeled by the number of clusters, followed by a comma, followed by the distance metric used, which is either cosine, square Euclidean, or Spearman rank correlation (denoted by cos, sqeuc, and spearman, respectively). The first bar assumes no clustering, i.e., a metric is not required. It is clear that clustering is beneficial as k increases from 2 to 16. Clearly, as k increases, energy reductions also increase. $k = 8$ results in more or equal energy reduction in comparison to 16 clusters.

Although the GM of the write energy of the proposed solution is measured across all the 32 applications, the training set, which is used to generate the k FVE mappings, is only composed of 16 applications. Fig. 4.6 compares the GM energy of the proposed solution across all applications, training set applications, and evaluation set applications. Energy reduction across the evaluation set for $k = 1$ is almost equivalent to DCW. As k increases, the reduction across the evaluation set improves. Although energy reduction across all the applications for $k = 8$ is better than $k = 16$, the energy reduction across the evaluation sets is marginally better ($\approx 1\%$) for $k = 16$ in comparison to $k = 8$.

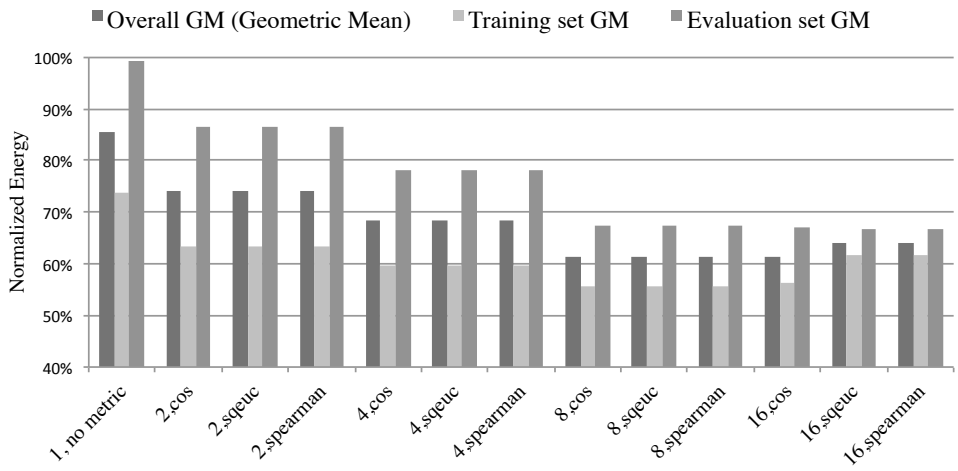

Figure 4.6: Each group of three bars represent the overall geometric mean (GM), training set GM, and evaluation set GM of MLC write energies. Similar to Fig. 4.5, write energies are normalized to DCW. As the number of clusters increases, the gap between write energies of the training set and evaluation set reduces.

Fig. 4.7 is the TLC equivalent of Fig. 4.5. The methods from left to right are: DCW, MFNW for TLC, incomplete data mapping (IDM) [Niu *et al.* 2013], and perfect knowledge FVE. NVM overheads of these methods are 0%, 1.8%, 20%, and 0%, respectively. Similar to the MLC case, as k increases, the energy reduction also improves. To compare energy reductions across the training and evaluation sets, we also provide a break down of this result in Fig. 4.8. Similar to MLC, when no clustering is performed ($k = 1$), energy reduction is only 5% better than DCW across the evaluation set. As k increases, the energy reduction across evaluation set improves. NVM overheads of the proposed solution are 1.8% for $k \leq 4$, and 3.5% for $k = 8$ and 16.

In all the previous charts, we divide the 32 applications into two halves: training set and evaluation set. To show the robustness of the clustering, we re-run the simulations using a training set size of 8 applications and evaluation set size of 24 applications. Since we use only 8 benchmarks for training, a cluster size of 16 is no longer possible, and therefore, we only show the result for k =2, 4, and 8. Fig. 4.9 shows the robustness chart in the MLC case. Clearly, the clustering is robust and the energy of the 25% training set size case (8 applications) is only 3.34% more on average in comparison to the 50% training set size (16 applications). The result is similar in the TLC case (Fig. 4.10), as the average energy is only 3.5% more.

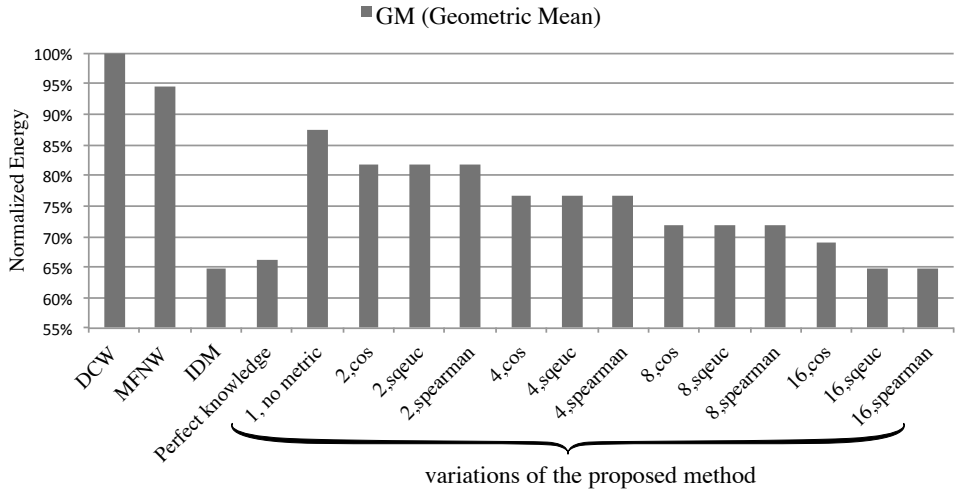

Figure 4.7: Results, organized similar to Fig. 4.5, for state-of-the-art TLC encoding solutions (left to right): MFNW for TLC, incomplete data mapping (IDM) [Niu *et al.* 2013], and perfect knowledge FVE. Again, write energies are normalized to DCW.

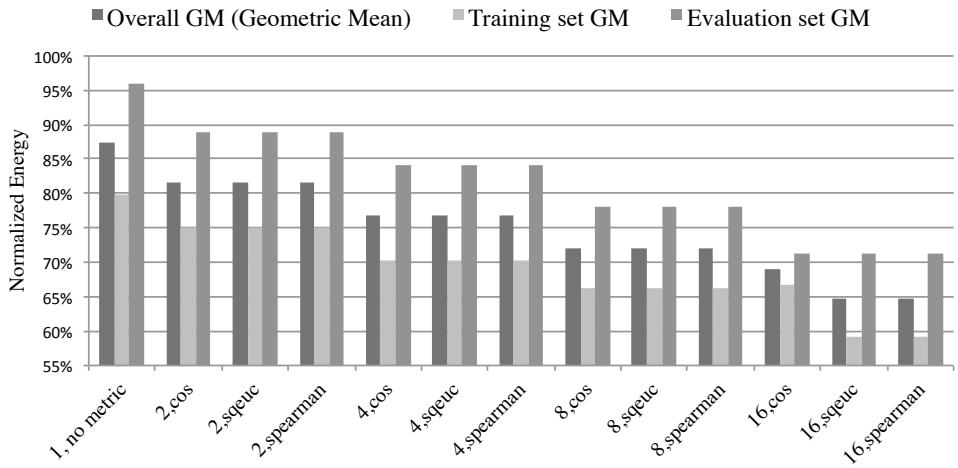

Figure 4.8: Results for TLC RRAM, organized similar to Fig. 4.6. The gap between write energies of the training and evaluation sets consistently decreases as k increases, except for the case from $k = 8$ to $k = 16$ for the square Euclidean and Spearman metrics (12% in both cases).

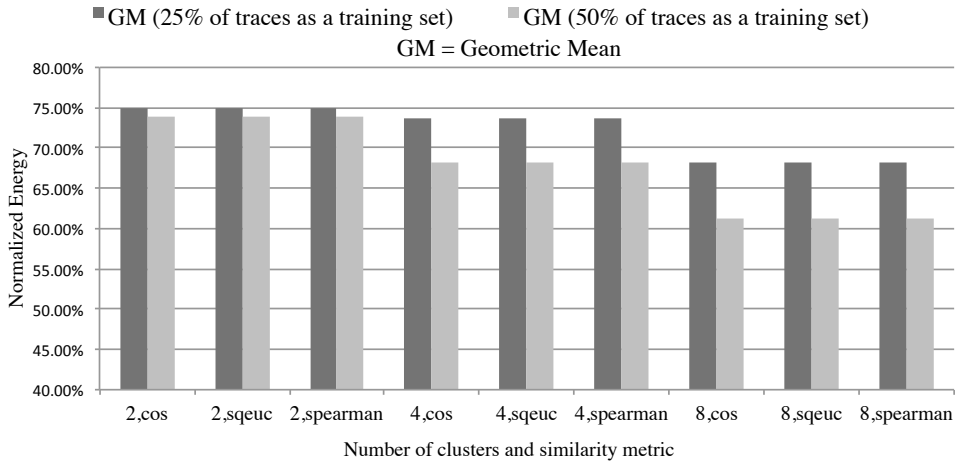

Figure 4.9: Geometric mean (GM) of MLC write energies, normalized to DCW. The two cases are training on 50% of the applications and evaluation on 50% versus training on 25% and evaluation on 75%. By training on 25% of the applications, write energy increases by only 3.34% on average.

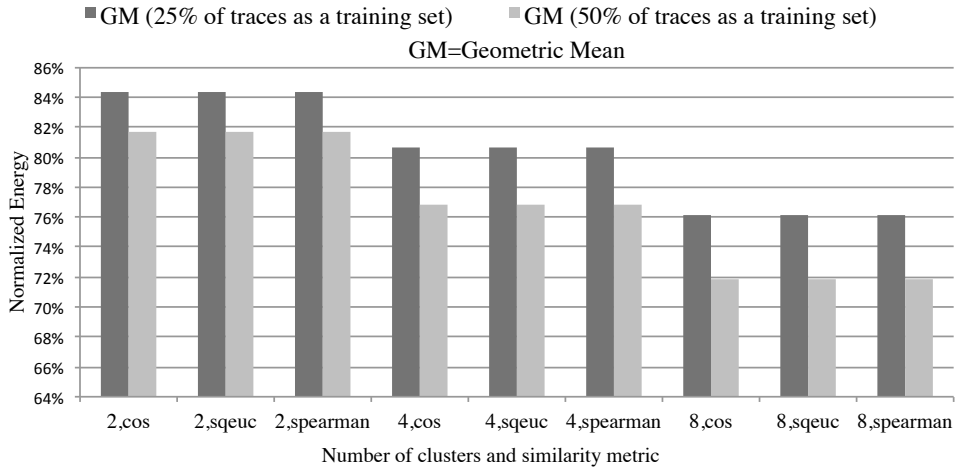

Figure 4.10: TLC case similar to Fig. 4.9. By training on 25% of the applications, write energy increases by only 3.5% on average in comparison to training on 50% of the applications.

In Fig. 4.5, the 50% overhead method produces the lowest MLC write energy, which is almost identical to perfect knowledge, but costs 50% NVM overhead, which is about 16× more than our overhead. Moreover, the average write energy of the proposed solution is only 5% more than that of perfect knowledge when using 8 clusters. In comparison to MFNW and cell remapping, the proposed method consumes 29% and 24% lower energy, respectively.

For the TLC result in Fig. 4.7, IDM results in marginally better energy reduction over perfect knowledge ($\approx 1\%$). Note that our proposed solution is comparable to IDM when $k = 16$ for the square Euclidean and Spearman metrics. However, IDM has NVM overhead of $20\% \approx 5.7\times$ the proposed solution NVM overhead.

4.3 CONCLUSIONS

This chapter proposed an energy efficient, low overhead offline FVE for MLC/TLC NVMs that approaches the performance of perfect knowledge FVE. The main idea is to cluster the perfect knowledge frequency profiles of a broad set of general-purpose applications to generate combined FVE mappings for each cluster. Results show that the average write energy for MLC PCM of the proposed method is only 5% more than the average write energy of perfect knowledge FVE. For TLC RRAM, the average write energy of the proposed solution is better by 1%. Furthermore, the low NVM and codec overheads makes the proposed method easy to implement and integrate into modern memory controllers.

5.0 L^{3}EP: A LOW LATENCY, LOW ENERGY PROGRAM-AND-VERIFY APPROACH

Data encoding solutions have been proposed to mitigate the side effects of P&V approaches on MLC/TLC PCM by reducing the mean energy and (in some cases) the mean latency [Mirhoseini *et al.* 2015, Wang *et al.* 2011]. Whereas encoding solutions focus on reducing the average write energy (latency), they cannot reduce the maximum write energy (latency); hence, there is strong motivation to develop low latency, low energy P&V solutions to accelerate the feasibility and commercialization of MLC/TLC PCM technology.

In what follows, we list the contributions presented in this chapter. First, we present L^{3}EP, a low latency, low energy P&V solution for TLC PCM (Section 5.1.1). L^{3}EP utilizes a multiple linear regression model to reach the target TLC state in just one (at most five) pulse(s) for 53% (>95%) of our comprehensive Monte Carlo (MC) simulations. L^{3}EP also accelerates the naturally slow PCM crystallization process by packing crystallization pulses separated by short idle periods, and augmenting them with a single verify step. L^{3}EP thus avoids unnecessary P&V verification steps resulting in faster crystallization over state-of-the-art P&V approaches. Second, we describe a comprehensive framework to compute critical L^{3}EP P&V-related parameters for multiple technology nodes (Section 5.1.2). The framework uses a series of MC simulations and optimizations imposing bounds on those parameters for the target technology node. Although this work focuses on TLC PCM, it can also be applied to MLC PCM through appropriate write margin and resistance ranges parameterization, as explained in Section 5.1.2.

We compare L^{3}EP to three state-of-the-art TLC PCM P&V approaches: (i) staircase up programming (SCUP) [Bedeschi *et al.* 2009], (ii) proportional-integral-derivative programming (PIDP) [Papandreou *et al.* 2011], and (iii) dual-pulse programming [He *et al.* 2014] (an amorphization-only P&V approach referred to as AOP henceforth). Results (Section 5.2) indicate that L^{3}EP can reduce the mean latency (energy) by 2.4–15× (1.9–12.2×) in comparison to SCUP, PIDP,

and AOP. Moreover, L^{3}EP reduces the worst case latency (energy) by 2.8–9.1× (2.1–11.4×) in comparison to SCUP, PIDP, and AOP. Despite this significant improvement in the programming latency, energy, and endurance of TLC PCM, the hardware overhead of L^{3}EP is comparable to SCUP, PIDP, and AOP.

The rest of this chapter is organized as follows. The main contributions pertaining to L^{3}EP are described in Section 5.1. Results are presented in Section 5.2. Finally, Section 5.3 concludes this chapter.

5.1 CONTRIBUTIONS

This section describes L^{3}EP, a low latency, low energy P&V solution for TLC PCM and the comprehensive customization framework for parameter optimization and technology scaling of L^{3}EP.

5.1.1 L^{3}EP

L^{3}EP is a P&V algorithm that uses amorphization and crystallization programming pulses to program TLC PCM. However, L^{3}EP favors amorphization pulses for programming speed and only resorts to the slow crystallization when fine–tuning is necessary. Similar to PIDP, L^{3}EP does not follow a specific P&V strategy and uses both crystallization and amorphization pulses as required. The rest of this section uses the L^{3}EP flowchart from Fig. 5.1 to describe the L^{3}EP P&V methodology.

First L^{3}EP iteration: Fig. 5.1 shows the flowchart of L^{3}EP. An iteration in L^{3}EP starts from initializing the crystallization pulse counter (C), where L^{3}EP iteration refers to the start of a TLC programming; the discussion of the role of the crystallization pulse counter is deferred to Section 5.1.1.2. After initializing C, L^{3}EP senses the resistance of the cell (R) and computes the programming error $E = R - R_M$, where R_M is the midpoint resistance of the target TLC state. If $|E| \leq \epsilon$, where ϵ is the write margin, the target TLC state is reached and L^{3}EP concludes. If $|E| > \epsilon$, L^{3}EP has to decide between amorphization and crystallization. If $E < -\epsilon$, crystallization is not an option, since crystallizing the PCM cell in this case will only make E more negative. If

$E > \epsilon$, however, L^{3}EP may use either crystallization or amorphization; however, since the crystallization process is naturally slow, it is only used when $\epsilon < E < \alpha\epsilon$, where α is L^{3}EP amorphization control parameter. In general, lower values of α further restrict the usage of crystallization, and thus can further speedup programming. However, to guarantee convergence to the target TLC state within a reasonable time, α has to be ≥ 2 to enable granular resistance tuning when the cell resistance is 2ϵ away from the target state. However, since closely spaced amorphization pulses can potentially overheat the PCM cell, L^{3}EP increases the value of α after every amorphization pulse. This ensures that on subsequent programming cycles, amorphization will only occur if $E < -\epsilon$.

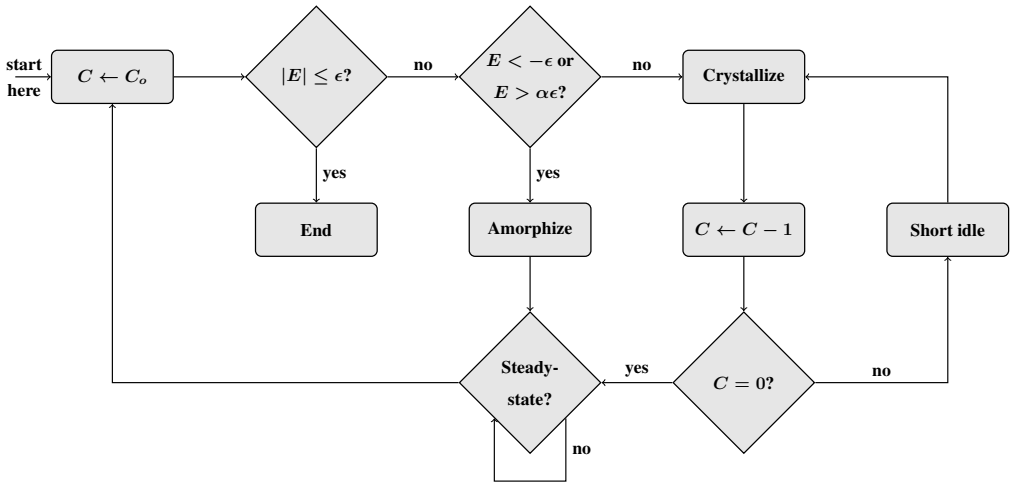

Figure 5.1: L^{3}EP flowchart: L^{3}EP starts by initializing the crystallization pulse counter (C). Then, it calculates the programming error and, based on the magnitude and sign of the error, it either stops, amorphizes, or crystallizes. Note that the first amorphization pulse amplitude is calculated using the linear regression model. Subsequent amplitudes are calculated incrementally as a function of the error. L^{3}EP crystallization is repeated after a short idle if $C > 0$, otherwise L^{3}EP waits for the steady-state resistance and repeats until the error is within the write margin.

5.1.1.1 L^{3}EP amorphization regression model Whereas a full amorphization pulse transforms the PCM cell to its highest resistance [Burr *et al.* 2010, Bedeschi *et al.* 2009], transforming the cell to arbitrary resistance levels lower than the highest value is possible using partial amorphization pulses, as shown in [Papandreou *et al.* 2011, He *et al.* 2014]. L^{3}EP leverages the idea of partial amorphization if $E < -\epsilon$ or $E > \alpha\epsilon$. When partial amorphization is chosen, the following linear regression model is used to predict the amplitude of the first amorphization pulse (V):

$$V = \sum_{i=0}^{n} \beta_i \left(R_M\right)^i . \tag{5.1}$$

Here, $\beta_0, \beta_1, \ldots$ are the model parameters, n is the degree of the predictor, and $(R_M)^i$ is the midpoint of the target resistance range raised to the i^{th} power. In practice, a linear predictor ($n = 1$) combines good prediction with affordable computation. In contrast, a cubic predictor ($n = 3$) provides better prediction quality at the expense of computational overhead. Note that the quadratic predictor ($n = 2$) is non-monotonic in the interval of interest, and is excluded from our evaluations. Further, the goodness of fit of predictors with $n \geq 4$ is marginal in comparison to the cubic predictor as indicated in Table 5.1, even as the computational overhead increases linearly with n. Therefore, we rule out predictors with $n \geq 4$ from our evaluations, and evaluate L^{3}EP using the linear and cubic predictors in this work.

Table 5.1: Goodness of fits and computational overheads (number of ADD/MULT for polynomial evaluation) for $\boldsymbol{n}$-degree polynomials, $\mathbf{1} \leq \boldsymbol{n} \leq \mathbf{6}$.

n	1	2	3	4	5	6
R-squared	57.51%	71.61%	72.15%	72.34%	73.19%	73.62%
RSE (MΩ)	1.11	0.91	0.90	0.90	0.89	0.88
Comp. overhead	2	4	6	8	10	12

In order to optimize β_i parameters in Eq. 5.1, we construct the following dataset. To account for manufacturing variability, we run Monte Carlo (MC) simulations, changing the PCM cell dimensions and the threshold voltage of the access device on every iteration within the variability limits. On every iteration, we apply a fixed width amorphization pulse at the gate terminal of the access device of the PCM cell. We assume the amplitude of the pulse (V) and initial cell resistance of the PCM cell (R) are uniformly distributed across MC iterations. On every iteration, after an application of a pulse, we measure the steady-state resistance (R_{ss}). The dataset is composed of the pairs (R_{ss}, V) generated after each MC iteration. We use these pairs to fit the parameters β_i using the least squares method [Chatterjee and Hadi 1986, N. and Smith 1981].

Once V is determined by Eq. 5.1, L^{3}EP issues the programming pulse, waits for the steady-state resistance, and starts a new iteration.

Subsequent L^{3}EP iterations: From the second iteration, L^{3}EP uses a different a different approach to update the pulse amplitude during the "Amorphize" step as follows.

$$V(t) = V(t-1) + M\Delta V. \tag{5.2}$$

Here, t is the time step, ΔV is the amount of update, and M is the update multiplier. The amount of the update, ΔV, is proportional to E. The update multiplier, M, is used to ensure that the step size is scaled sufficiently to effect a measurable change in the cell resistance. Initially, M is set to 1. On subsequent iterations, if the cell resistance changes only slightly in response to the last pulse amplitude, M is increased to ensure that the cell resistance changes in the desired direction. In other words, the update multiplier, M, is 1 unless the difference between the previous and current resistances is negligible, in which case L^{3}EP sets M to 10. For simplicity, L^{3}EP operates with M set to 1 or 10; $M = 10$ is chosen such that $V(t)$ stays within its permissible range.

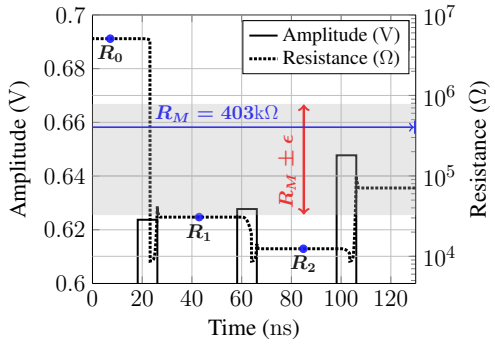

Figure 5.2: L^{3}EP amorphization based on Example 1.

Example 1: Fig. 5.2 illustrates successive updates on the pulse amplitude to reach TLC state 6. Without loss of generality, this example assumes a 32 nm PCM cell with dimensions summarized in Table 5.2. Initial α is 2.2, $\epsilon = 370\text{K}\Omega$, and $R_M = 403\text{K}\Omega$ (TLC state 6). The plot sketches the programming pulse amplitude (left y-axis) and the PCM cell resistance in logarithmic scale (right y-axis), versus time. Also, the upper and lower bounds of the write margin of TLC state 6 are shown. The first pulse amplitude is calculated using Eq. 5.1. The second and third amplitudes are calculated using Eq. 5.2 with $M = 1$ and $M = 10$, respectively. We set $M = 10$ in the third pulse since $|R_1 - R_2|$ is relatively small, where R_1 and R_2 are the steady-state resistances after the application of the first and second pulses, respectively.

5.1.1.2 L^{3}EP crystallization L^{3}EP crystallization is chosen if $\epsilon < E < \alpha\epsilon$. L^{3}EP packs a number of crystallization pulses separated by short idle periods. The short idle periods: (i) prevent the cell from reaching the full crystalline state and (ii) reduce the crystallization time constant of the second and subsequent pulses. Further, since L^{3}EP does not wait for the steady-state resistance or check the programming error between the packed pulses, it drastically reduces the duration for crystallization programming. The number of packed pulses, i.e., the crystallization pulse counter (C) in Fig. 5.1, is proportional to $|E|$; in this work, 2–3 crystallization pulses are packed unless $|E|$ is small, when a normal single crystallization pulse is used. The amplitude of the crystallization pulse is proportional to $|E|$ and is incremented by a fixed ΔV.

L^{3}EP implements pulse packing as follows. The crystallization pulse counter, C, is initialized to C_o, i.e., the number of pulses to be packed. After every crystallization pulse, L^{3}EP decrements C, and if $C > 0$, L^{3}EP waits for a short idle period before initiating crystallization again. If $C = 0$, L^{3}EP waits for the steady-state resistance and returns to the initial step to start a new iteration.

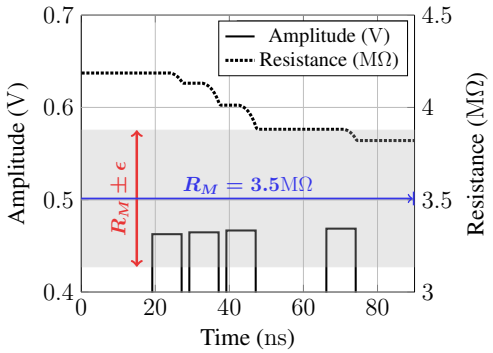

Figure 5.3: L^{3}EP crystallization based on Example 2.

Example 2: Fig. 5.3 illustrates crystallization packing to reach TLC state 2. Without loss of generality, this example assumes a 32 nm PCM cell with dimensions and parameters summarized in Table 5.2, α is initially 2.2, and $\epsilon = 370$KΩ. Moreover, this example also assumes that the midpoint resistance of the target TLC state is 3.5MΩ. Three crystallization pulses, separated by 2 ns, are packed at $t \approx 19$ ns. Note that although the first pulse barely affects the resistance, the complete pack of pulses reduces the resistance by $\approx$ 300 K Ω. Next, L^{3}EP waits for the steady-state resistance and starts a new iteration, wherein it estimates that a single crystallization pulse is required to reach TLC state 2. In this example,

pulse packing saves (i) 2× the wait duration to the steady-state resistance and (ii) 2× the duration of the error verification step.

5.1.2 Parameter Optimization

This subsection describes the computation and technology optimization of L^{3}EP P&V parameters. The parameters are (a) the minimum crystallization and amorphization voltages V_c and V_m, (b) amplitudes and durations of full SET and RESET pulses, (c) resistance ranges of TLC states, (d) write margin ϵ, and (e) time to the steady-state resistance after the application of a programming pulse [Burr *et al.* 2010, Bedeschi *et al.* 2009, Papandreou *et al.* 2011]. Without loss of generality, we assume a voltage-controlled [Braga *et al.* 2010] 1T/1R PCM cell (Fig 5.4) with L^{3}EP P&V controller connected to BL and WL connected to V_{DD}. Technology optimization parameters include access device and PCM cell parameters. To account for manufacturing variability, the dimensions of the PCM cell and threshold voltage of the access device are drawn from normal distributions with appropriate standard deviations.

Minimum V_c and V_m: These are the pulse amplitudes that raise the GST temperature (T) just above the crystallization and melting temperatures, respectively, assuming a suitable pulse width. To the first order, the pulse width is directly proportional to the chosen feature size. Since the GST temperature, T, is one of the outputs exposed by the PCM model used in this work [Xu *et al.* 2012], we setup the following simulation to determine V_c. While the access device is turned on, a small–amplitude pulse is issued at the BL terminal of the 1T1R circuit (Fig. 5.4). We gradually increase the amplitude of the pulse until T is just above the crystallization temperature. At this point, the amplitude becomes a candidate value of V_c. To account for variability, we run Monte Carlo (MC) simulations, changing the PCM cell dimensions and access device threshold voltage on every iteration, to determine the minimum V_c. The minimum amorphization voltage (V_m) is determined similarly.

Full SET and RESET pulses: Since the full SET pulse is supposed to fully crystallize the PCM cell, the amplitude of the pulse is $\geq V_c$ and $< V_m$. We pick a value in this range and sweep the duration of the pulse until the PCM cell is fully crystallized. There is a latency–energy tradeoff

between the duration and amplitude of the pulse. Small pulse amplitudes may reduce energy but possibly increase latency and vice versa. Once a duration and amplitude are chosen, we perform MC simulation to ensure that the cell switches to the full crystalline phase in all MC iterations. The upper bound on the full crystalline state resistance, R_7, is the maximum resistance across all MC iterations. The full RESET pulse and R_0 (the lower bound on the resistance of the full amorphous state) are determined in a similar manner.

Resistance ranges: Once the lower and upper bound resistances (R_0 and R_7) of the full amorphous and crystalline states, respectively, are determined, we can determine the linear-spaced resistance ranges of the intermediate TLC states as follows:

$$R_i = R_7 + (6 - i)(R_0 - R_7)/6.$$

Here, R_i corresponds to the lower bound resistance of TLC state i and $0 < i < 7$. Therefore, the midpoint resistance of intermediate TLC state i is $R_{M_i} = (R_i + R_{i+1})/2$. This results in a write margin, $0 < \epsilon < |R_{i+1} - R_i|/2$, where $0 < i < 7$. Using the full bandwidth of the cell resistance may reduce the expected latency, since it results in larger ranges, and in turn increases the probability of reaching the target TLC state within the first few iterations. Alternatively, we can use a restricted resistance range to reduce the average pulse amplitude and hence reduce the expected energy. However, with a restricted resistance range, it is not possible to program TLC state 0 using a single full RESET pulse, implying an increase in the expected latency and energy for TLC state 0. Moreover, full and restricted resistance ranges have the same reliability issues, e.g., both need a drift tolerance methodology, e.g., [Awasthi *et al.* 2012], and both may suffer from retention failures [Burr *et al.* 2010]. Therefore, we adopt the full resistance range in this work.

Write margin (ϵ): This parameter is used to decide programming continuation/termination conditions. As stated above, ϵ can be as large as $|R_{i+1} - R_i|/2$ for $0 < i < 7$. Note that the write margin must not be confused with the sense/read margin. For example, the write margin of state 3 is between $R_{M_3} - \epsilon$ and $R_{M_3} + \epsilon$ and the read margin is between R_3 and R_2. This implies that lower values of ϵ can lead to a more drift-tolerant P&V.

Time to the steady-state resistance (T_{ss}): T_{ss} is determined after the application of a short pulse by observing the GST temperature. T_{ss} equals the time duration between the falling edge of the

pulse and the point at which the temperature drops just below the GST crystallization temperature [Xu *et al.* 2012]. To account for variability, we perform MC simulations and vary the pulse amplitude, PCM cell dimensions, and access device threshold voltage on each MC iteration. The maximum T_{ss} across all MC iterations, rounded up to the nearest multiple of the memory controller's clock cycle, is the final value of T_{ss}. However, we emphasize that T_{ss} is only used after crystallization pulses. After an amorphization pulse, the next pulse has to wait for about 30 ns [Burr *et al.* 2010], i.e., the time till a melted PCM cell is fully back to the solid phase.

5.2 RESULTS

This section presents the results of the cell-level and full system simulations of L^{3}EP. It also discusses the hardware overhead of L^{3}EP in comparison to SCUP, PIDP, and AOP.

5.2.1 Cell-Level Evaluation

In this section, we first describe our simulation environment and configuration, and discuss the linear regression model (Eq. 5.1). Next, we present the latency, energy, and endurance results of L^{3}EP in comparison to SCUP, PIDP, and AOP.

Simulation setup: In this work, we extend the Verilog–A compact model for the PCM cell proposed in [Xu *et al.* 2012]. To account for variability, we perform extensive Monte Carlo (MC) simulations changing the PCM cell and access device parameters on every iteration. Throughout this section, we assume a 32 nm PCM cell, whose dimensions and access device threshold voltage are normally distributed with a variability of $\mu \pm 3\sigma$, where the means and standard deviations are summarized in Table 5.2. All the P&V approaches evaluated in this work are integrated into the 1T1R model [Xu *et al.* 2012] (Fig. 5.4). To estimate the expected programming latency, without loss of generality, we assume that the resistance of the PCM cell can be sampled in 10 ns, as reported in [Burr *et al.* 2010]. We assume a GRFPU [Catovic 2004] floating point unit with a controller clock period of 1 ns. We also choose a pulse width of 8 ns and a dual-pulse width of 2×8 ns, separated by 2 ns. These assumptions allow us to estimate the latency of the proposed and state-of-

the-art P&V approaches. Furthermore, we evaluate every P&V approach using two values of the write margin (ϵ), i.e., 370KΩ and 150KΩ. The larger ϵ results in lower latency, whereas the smaller ϵ results in larger separation between the write margins of TLC states, making the cell more tolerant to resistance drift. In other words, with a smaller ϵ, it takes longer for the resistance to drift and cause a soft error. Nevertheless, like all P&V approaches for MLC/TLC PCM, L^{3}EP requires a separate drift tolerance mechanism, e.g., data scrubbing [Awasthi *et al.* 2012]. Finally, note that we neglect the timing of analog-to-digital and digital-to-analog conversions (ADC/DAC), as state-of-the-art ADC/DAC have relatively negligible latencies (15–50 ps) [Laperle and O'Sullivan 2014].

Table 5.2: (a) PCM cell and (b) access device parameters. We use a 32 nm high power predictive technology model nMOS [ptm] with $\boldsymbol{W/L = 4}$ to provide enough current.

(a) *PCM cell parameters*

Parameter	Value
Contact width	28 nm $\pm$ 4%
PCM thickness	49 nm $\pm$ 2%
V_{DD}	0.9V

(b) *nMOS access device parameters*

Parameter	Value
Length	28 nm
Width	128 nm
V_T	130 mV $\pm$ 10%

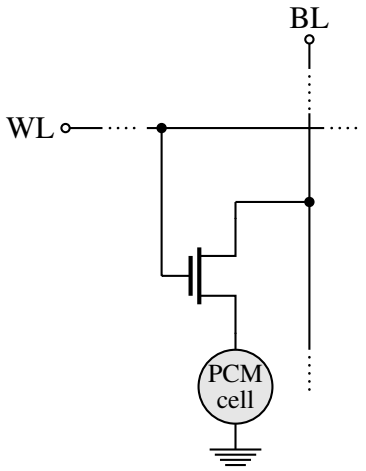

Figure 5.4: A 1T1R PCM cell with an nMOS access device: The PCM cell can be controlled using the bit line (BL) and word line (WL) inputs.

Regression analysis: Fig. 5.5 shows the result of the regression analysis of the amplitude of the first amorphization pulse. The x-axis is the steady-state resistance and the y-axis is the pulse amplitude. The scatter points in the plot represent the regression dataset. As explained previously,

L^3EP is evaluated with the linear and cubic predictors (as seen, the quadratic predictor is non-monotonic). Whereas the cubic predictor is more accurate than the linear predictor, it requires more computational effort. Note that linear prediction requires a multiplication and an addition, whereas cubic prediction requires three additions and three multiplications. However, if this overhead is acceptable, the cubic predictor is preferable, since it is more likely to reach the target TLC state in a single pulse. In the rest of this section, we refer to L^3EP with linear and cubic predictors as L^3EP (1) and $L^3EP(3)$, respectively.

Terminal TLC states: We assume that all the P&V approaches considered in this work, including L^3EP, perform regular SLC programming for the terminal TLC states 0 and 7. Therefore, we exclude these two TLC states from our comparisons since they behave identically across all the evaluated P&V methods. However, for completeness, we report the result of MC simulations of these two states. TLC states 0 and 7 have latencies of 15 ns and 35 ns, with mean energies of 0.52 pJ and 0.41 pJ, respectively. Also, TLC state 7 has a maximum resistance of R_7 =15KΩ and state 0 has a minimum resistance of R_0 =4.67MΩ.

MC simulations: Every P&V approach considered in this work is simulated for 12K MC iterations, 6K per value of ϵ. On every MC simulation, the PCM cell is programmed to every intermediate TLC state for $\approx$ 1K iterations. Therefore, the total number of MC iterations is 60K. On every iteration, the PCM cell dimensions and the nMOS threshold voltage are varied according to Table 5.2, and the initial cell resistance and target TLC state are assumed to be uniformly distributed across all MC iterations. The output of every MC iteration includes the PCM cell dimensions, the threshold voltage of the nMOS device, the initial resistance, the target TLC state, the programming latency, the energy, and the steady-state resistance. Fig. 5.6 shows a stacked histogram with normalized frequency of the steady-state resistances, broken down by ϵ, then by P&V approach. The dashed vertical lines mark the lower and upper bounds (read margins) of every intermediate TLC state.

Latency: Whereas mean and standard deviation statistics provide good insight about data, they are likely to be influenced by outliers and should be explained in the light of underlying distribution. For this reason, we use violin plots [Hintze and Nelson 1998], which in addition to showing the spread of the data, also show the probability density of the data at every value; this creates variable-

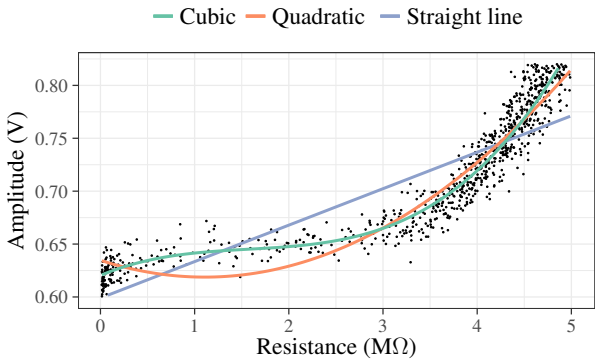

Figure 5.5: Regression plot of the steady-state PCM cell resistance (x-axis) versus the amplitude of the first amorphization pulse (y-axis). The scatter graph is the regression dataset. Linear, quadratic, and cubic models are shown. We assume a 32 nm PCM cell and access device with parameters from Table 5.2.

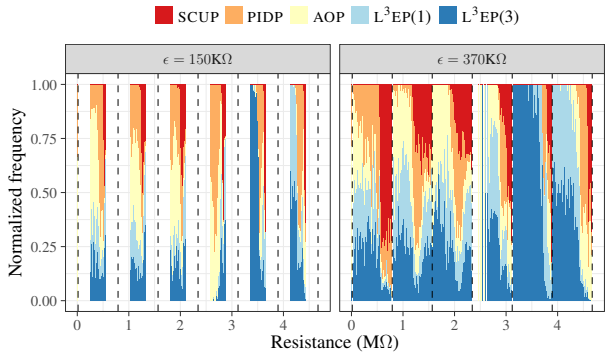

Figure 5.6: Resistance distributions of intermediate TLC states. Here, we show the steady-state resistance histograms for the two write margins: $\epsilon = 150\text{K}\Omega$ and $\epsilon = 370\text{K}\Omega$. The histograms result from a total of 60K MC iterations. The vertical dashed lines mark upper and lower bounds of read margins of intermediate TLC states, starting from TLC state 6 (left-most) to TLC state 1 (right-most). We assume a 32 nm PCM cell and access device with parameters from Table 5.2.

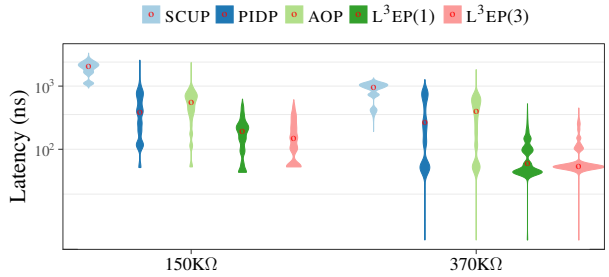

Figure 5.7: Write latencies broken down by the two write margins then by various P&V approaches. The y-axis is in logarithmic scale. The median latencies are indicated by ○ marks on every violin.

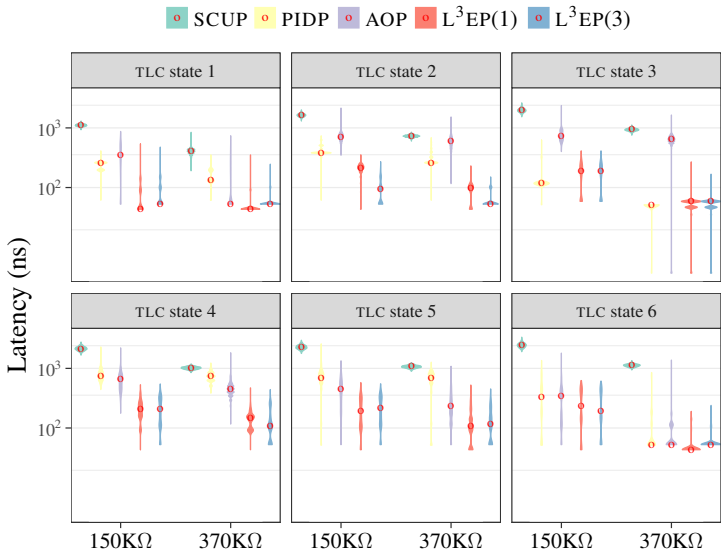

Figure 5.8: Write latencies broken down by TLC state, then by the two write margins, and then by various P&V approaches. The median latencies are indicated by ○ marks on every violin.

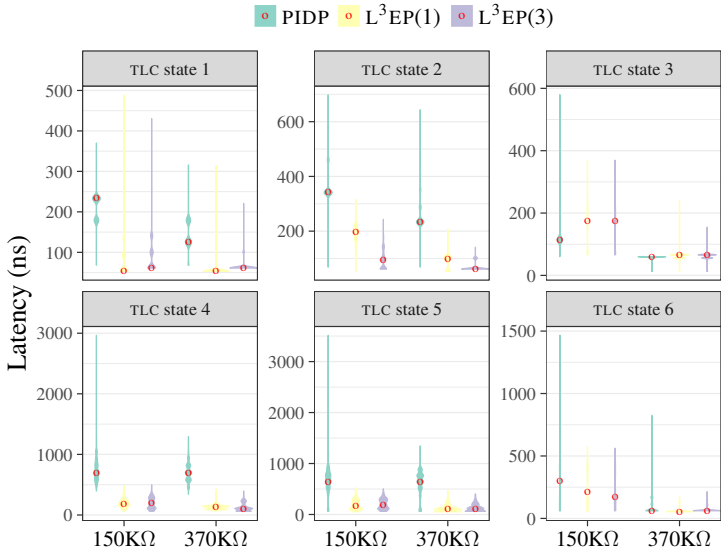

Figure 5.9: Similar to Fig. 5.8, but compares PIDP, L^{3}EP(1), and L^{3}EP(3). Also, the y–axis is linearly scaled in this figure. The median latencies are indicated by ○ marks on every violin.

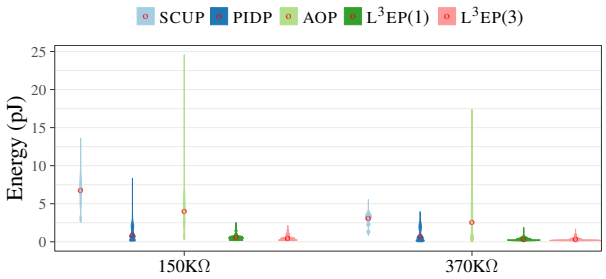

Figure 5.10: Write energies broken down by the two write margins then by various P&V approaches. The median energies are indicated by ○ marks on every violin.

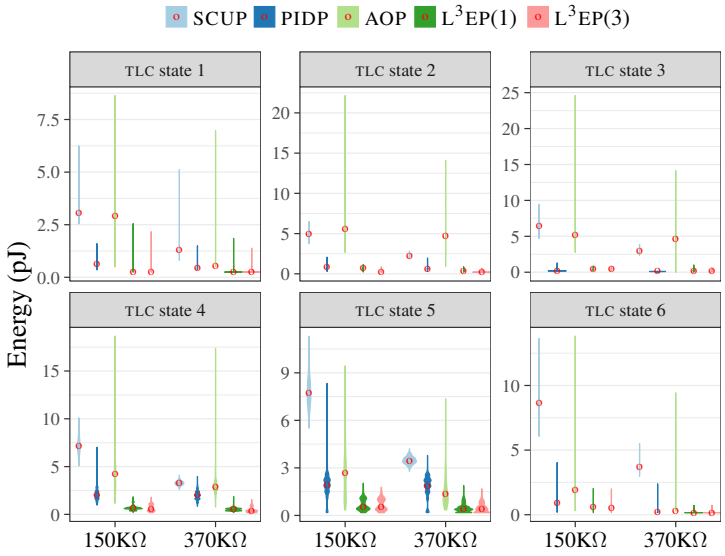

Figure 5.11: Write energies broken down by TLC state, then by the two write margins, and then by various P&V approaches. The median energies are indicated by ○ marks on every violin.

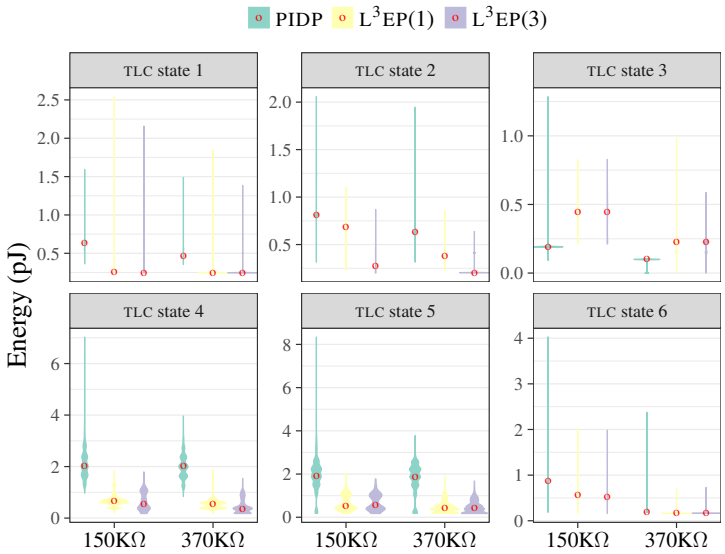

Figure 5.12: Similar to Fig. 5.11, but compares PIDP, L^{3}EP(1), and L^{3}EP(3). The median latencies are indicated by ○ marks on every violin.

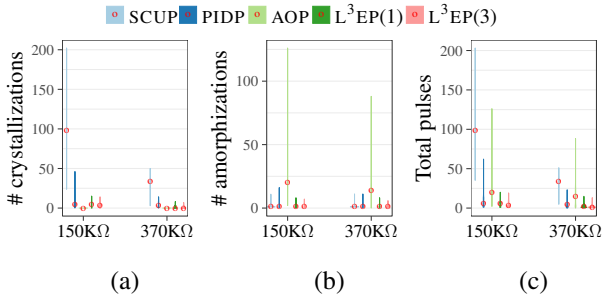

Figure 5.13: These plots show the number of (a) crystallization, (b) amorphization, and (c) total pulses of the 60K MC iterations broken down by the two write margins and then by various P&V approaches.

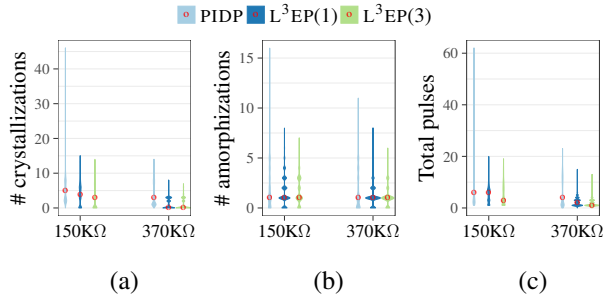

Figure 5.14: Similar to Fig. 5.13 but compares PIDP, L^{3}EP(1), and L^{3}EP(3) only.

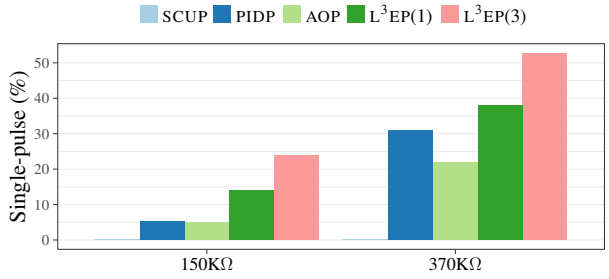

Figure 5.15: The chart shows the percentage of MC iterations in which the programming converges in one pulse. The x-axes group the results by ϵ, then by P&V approach.

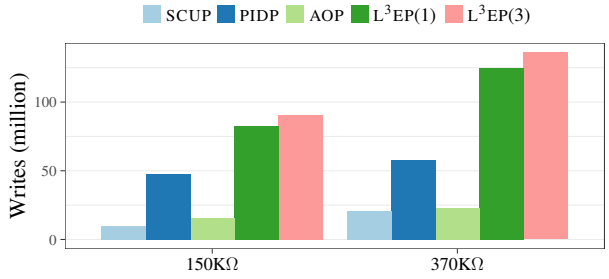

Figure 5.16: The chart shows the expected lifetime of a TLC PCM (in million writes). The x-axes group the results by ϵ, then by P&V approach.

width boxes (violins) for every evaluated P&V approach. Fig. 5.7 shows the latency violin plot aggregated across all intermediate TLC states. The x-axis breaks down the result by the write margin (ϵ), and then by P&V approach. The y-axis is the programming latency on a logarithmic scale (ns). For both values of ϵ, the latency distributions of L^{3}EP(1) and L^{3}EP(3) have smaller means and spreads in comparison to state-of-the-art P&V approaches, indicating correspondingly smaller standard deviations. P&V approaches are affected by ϵ in two ways: (i) the worst case latency increases when ϵ decreases and (ii) the probability densities of lower latencies are squeezed upwards and re-distributed among other higher latencies. Nonetheless, for both values of ϵ, the worst case latency of L^{3}EP is 2.8–9.1$\times$ lower than SCUP, PIDP, and AOP.

Fig. 5.8 breaks down the latency result further by intermediate TLC states. L^{3}EP(1) and L^{3}EP (3) have the lowest median latencies for TLC states 1, 2, 4, and 5. For TLC state 3, L^{3}EP(1) and L^{3}EP(3) have comparable median latencies to PIDP, which has the lowest median latency for this TLC state. For TLC state 6, all P&V approaches (except SCUP) have comparable median latencies.

Since the write latency varies at a large scale among L^{3}EP(1), L^{3}EP(3), AOP, PIDP, and SCUP, we show a similar plot as in Fig. 5.8 that only compares P&Vs with comparable latencies, i.e., L^{3}EP(1), L^{3}EP(3), and PIDP. The latencies of the new plot are linearly scaled as shown in Fig. 5.9. Again, L^{3}EP(1) and L^{3}EP(3) have the lowest median latencies for TLC states 1, 2, 4, and 5. For TLC states 3 and 6, PIDP latencies are comparable to L^{3}EP(1) and L^{3}EP(3). The best performance for PIDP occurs at TLC state 3 and $\epsilon = 370$, in which it excels in both the median and maximum write latencies, although both are comparable to L^{3}EP(1) and L^{3}EP(3).

Before we conclude this part, one final note about the large latency of SCUP for the case $\epsilon = 150\text{K}\Omega$. The reason for such large latency is that the programming pulse width is set to 6 ns. Choosing larger pulse width can speed up SCUP but does not guarantee convergence. This pulse width of 6 ns has been maximized as a result of a MC simulation that is described as follows. We start with a pulse width of 10 ns and run many MC iterations of SCUP. In case convergence fails one or more times, the pulse width is reduced by 1 ns and the MC simulation is repeated until no failures are observed. This exercise results in SCUP having a maximum pulse width of 6 ns. On the other hand, note that SCUP is much faster for $\epsilon = 370\text{K}\Omega$ since the maximum pulse width in this case is 8 ns.

Energy: Fig. 5.10 shows the violin plot for the write energy organized similar to Fig. 5.7, but the y-axis is linearly scaled (pJ). For both values of ϵ, the majority of L^{3}EP(1) and L^{3}EP(3) write energies are < 1 pJ, while the mean write energy is < 0.6 pJ for both L^{3}EP(1) and L^{3}EP(3). Moreover, for $\epsilon = 370\text{K}\Omega$, the mean write energy is < 0.35 pJ for L^{3}EP(1) and L^{3}EP(3). The worst case energy of L^{3}EP is at least $2.1\times$ lower than SCUP, PIDP, and AOP.

Fig. 5.11 breaks the write energy down by intermediate TLC states. L^{3}EP(1) and L^{3}EP(3) have the lowest median energy consumptions for TLC states 4 and 5. For TLC states 1, 2, 3, and 6 L^{3}EP(1), L^{3}EP(3), and PIDP have the lowest write energies and they seem to have very comparable energy consumptions; however, if we examine Fig. 5.12, which compares the energy consumptions of PIDP, L^{3}EP(1), and L^{3}EP(3), we see that PIDP is only superior at TLC state 3, in which it has a marginally lower median energy in comparison to L^{3}EP(1) and L^{3}EP(3). For the rest of the TLC states, L^{3}EP(1) and L^{3}EP(3) outperform PIDP (as well as AOP and SCUP).

Finally, note that even though AOP is expected to consume the highest energy since it only uses amorphization pulses, its median energy consumption is lower than SCUP for $\epsilon = 150\text{K}\Omega$ and for TLC states 3, 4, 5, and 6. This is because SCUP issues too many crystallization pulses to reach these TLC states causing its median energy consumption to exceed AOP. However, AOP can have very high energy consumption, e.g., up to 25 pJ for TLC state 3 and for $\epsilon = 150\text{K}\Omega$. We conclude from this that SCUP and AOP are not practical P&Vs with respect to energy consumption for the case $\epsilon = 150\text{K}\Omega$.

Number of pulses: Fig. 5.13 plots the number pulses issued by various P&V approaches and broken down by the pulse type, write margin, and P&V approach. First, note that SCUP (AOP) has the lowest (zero) amorphization (crystallization) pulse count because it primarily uses crystallization (amorphization) pulses to reach the target TLC state. On the other hand, SCUP (AOP) has the maximum crystallization (amorphization) pulse count. Fig. 5.14 better compares PIDP, L^{3}EP(1), and L^{3}EP(3) since it removes SCUP and AOP results from the plot, resulting in tighter y–axes. Although the median pulse counts of the three P&Vs are comparable, there is a big difference in the maximum number of pulses, especially for the case $\epsilon = 150\text{K}\Omega$. The number of total pulses of L^{3}EP(1) and L^{3}EP(3) are robust with respect to changing in ϵ, as they only slightly influenced by the change in ϵ. One final note, the median number of crystallization pulses for both L^{3}EP(1) and

L^3EP(3) is zero for $\epsilon = 370\text{K}\Omega$. In other words, in half of the MC iterations, L^3EP(1) and L^3EP(3) for $\epsilon = 370\text{K}\Omega$ converge using only amorphization pulses.

Effect of regression: Fig. 5.15 illustrates the effect of choosing a linear/cubic predictor by showing the number of instances when the programming converges in a single pulse. The x–axis is broken down by ϵ, then by P&V approach. The y–axis is the percentage of MC iterations that converge in a single pulse (or a single dual-pulse for AOP). Note that the number of instances that SCUP converges to intermediate TLC states within a single pulse is zero for both values of ϵ. This is because SCUP initializes the PCM cell before it starts the programming and thus always requires more than one pulse for intermediate TLC states. Also, note that for $\epsilon = 370\text{K}\Omega$, even PIDP (AOP) potentially converges within one pulse (dual-pulse), because the write margin is large. In this case, L^3EP(1) is only about 8% better than PIDP, and this is due to the fact that PIDP uses a linear actuator, but without regression (which accounts for the 8% improvement of L^3EP(1) over PIDP). However, for L^3EP(3), the increase over PIDP is about 21%. Although the drop in percentages when $\epsilon = 150\text{K}\Omega$ is expected (since ϵ is almost halved), L^3EP(1) and L^3EP(3) results in 2.7× and 4.6× improvement, respectively, over PIDP.

Endurance: SLC PCM cells have normally distributed lifetimes with a mean of 10^8 writes [Schechter *et al.* 2010]. Assuming a 32 nm SLC PCM, the mean energy per cell write is 0.465 pJ, and thus the expected total energy before the cell fails is $E_{\text{TOT}} = 46.5\mu\text{J}$. Whether a PCM cell operates as an SLC, MLC, or TLC, the cell is expected to consume an average of E_{TOT} pJ before failure. Therefore, if we assume that all TLC states have equal probabilities of occurrence and that a TLC write costs a mean energy of E_{TLC} pJ/write, we can estimate the expected number of TLC writes before failure by $E_{\text{TOT}}/E_{\text{TLC}}$ writes. Fig. 5.16 summarizes the lifetime findings as approximated using this simplified endurance model. L^3EP results in at least 1.7× lifetime increase in comparison to other P&V approaches. It is also worth noting that for $\epsilon = 370\text{K}\Omega$, L^3EP(3) is expected to improve endurance by 36% over pure SLC programming.

5.2.2 Full System Simulation

Simulation setup: We use gem5 [Binkert *et al.* 2011] and NVMain [Poremba and Xie 2012] for a cycle-accurate full system simulation. Gem5 combines the functionality of M5 [Binkert *et al.*

2006] and GEMS [Martin *et al.* 2005] simulators to provide a customizable full system simulation toolkit for a wide range of CPUs and ISAs. NVMain is a cycle accurate memory simulator that can simulate DRAM, NVMs, and hybrid memories. NVMain can be used as a standalone trace simulator or as a library that can be integrated seamlessly with full system simulators as gem5. We use gem5 and NVMain to simulate an x86 system whose parameters are shown in Table 5.3.

NVMain has an out-of-the-box MLC PCM modeling functionality. When simulating MLC PCM, it estimates the write latency of an MLC write request by generating a normally distributed random number that corresponds to the number of pulses of an assumed P&V approach. The number of pulses are then converted to time unit and the maximum latency is assumed for the write request. Write energies are estimated by multiplying the mean write energy per multi–level cell times the number of written cells. We modify NVMain in two ways: (i) we implement TLC PCM modeling and (ii) we model various P&V approaches for the two write margins, 150KΩ and 370KΩ. In order to accurately simulate the different P&V approaches, we model their latencies and energies at every TLC state. This is achieved by generating empirical cumulative distribution functions (ECDFs) of every latency and energy violin presented in Fig. 5.8 and Fig. 5.11. Every violin in these two figures corresponds to an empirical probability distribution function that can be easily converted to an ECDF.

Once the ECDFs of every TLC state are generated, we model the write latency of a write request as follows. The number of writes are counted for every TLC state. Then, for every TLC state, we generate a sample of the same size as the number of writes for that state. To generate a single latency sample, we use a roulette-wheel selection scheme [Lipowski and Lipowska 2011] as follows. A uniform random number $r \in [0,1]$ is generated and used as an input to evaluate the inverse ECDF, which outputs a latency l (the sample) whose cumulative probability is r. This procedure is repeated to generate more samples. After all the latency samples are generated, we use the maximum value of all the samples as our write latency for the write request. Fig. 5.17 demonstrates the similarity between the samples generated by the roulette-wheel scheme and the latencies samples as computed by SPICE. The figure compares ECDFs of the latencies calculated by SPICE and generated by the roulette-wheel scheme for the five P&Vs and for $\epsilon = 150\text{K}\Omega$. For brevity, we only show the ECDFs of TLC state 4.

In NVMain, it is assumed that there are enough drivers to simultaneously program all the cells of the entire memory row (512-TLC). Whereas limiting the number of drivers is followed in practice [Bedeschi *et al.* 2009], it only influences absolute key figures. However, in our case, we normalize all the results to SCUP for $\epsilon = 150\text{K}\Omega$. This normalization cancels out the difference between the two cases of unlimited and limited number of drivers. The following example illustrates the idea.

> **Example 3:** Suppose that the absolute latencies with unlimited drivers for SCUP and PIDP are l_{us} and l_{up}, respectively. If these latencies are normalized to l_{us}, then the normalized latencies for SCUP and PIDP are 1 and $l_{\text{us}}/l_{\text{up}}$, respectively. In the case of a limited number of drivers, say n drivers per memory array, the latencies of SCUP and PIDP are going to scale up by $\approx l_{\text{us}}N/n$ and $l_{\text{up}}N/n$, respectively, where N is thr number of cells in a memory line. If we normalize these latencies to $l_{\text{us}}N/n$, we observe the same normalized latencies as in the unlimited drivers case. Therefore, limiting the number of drivers is not important in a simulation if we use normalization. However, it is important in practice or if we need to calculate the latency in units of time.

Despite this fact, not only that we simulate the unlimited drivers case, but also, we simulate two more limited drivers cases. Therefore, our simulation cases consists of the following cases:

- Case (i): We assume that all evaluated P&Vs have unlimited drivers. The purpose of considering this case is to show the upper bound gain in performance for L^{3}EP in comparison to other P&Vs.
- Case (ii): We assume that SCUP has 16 drivers per memory line, whereas other evaluated P&Vs, including L^{3}EP, are assumed to have 4 drivers per memory line.
- Case (iii): We assume that SCUP has 16 drivers per memory line, whereas other evaluated P&Vs, including L^{3}EP, are assumed to have only 1 driver per memory line.

Clearly, our simulation assumptions in Cases (ii) and (iii) give SCUP advantage over the rest of the evaluated P&Vs in terms of the number of drivers. The reason for this is that 16 drivers is already a realistic assumption in the MLC case [Bedeschi *et al.* 2009], so we assume it is also possible in

the TLC case. We have not assumed the same number of drivers for the other P&Vs because their drivers cannot be shared among cells, like the case of SCUP.

For all these cases, and for every ϵ and P&V, we simulate the workloads shown in Table 5.4. Note that every workload is composed of four SPEC CPU2006 benchmarks to keep all the CPUs of the simulated system busy. Furthermore, for every workload, we bind every benchmark application to a certain CPU using `taskset` utility. This ensures the utilization of all the four CPUs of the simulated system. Table 5.5 shows L2 cache misses–per–kilo–instruction (MPKI) for every workload, P&V, and ϵ. For every workload, P&V, and ϵ, we use gem5 and NVMain to simulate 300 million instructions after fast-forwarding for 4 billion instructions.

Table 5.3: Full system specifications

CPU	Four OoO at 4GHz, each with 128KB instruction and data L1 caches
L2 cache	Shared 8–way 8MB with 512b cache line
Memory controller	FR-FCFS, open-page, bank first round-robin at 1GHz
Technology	PCM
Protocol	LPDDR, x8
t_{RCD}	120 ns [Poremba and Xie 2012]
Read energy	0.2 pJ/bit [Poremba and Xie 2012]
Memory organization	4 banks/rank, 2 ranks/channel, 2 channels

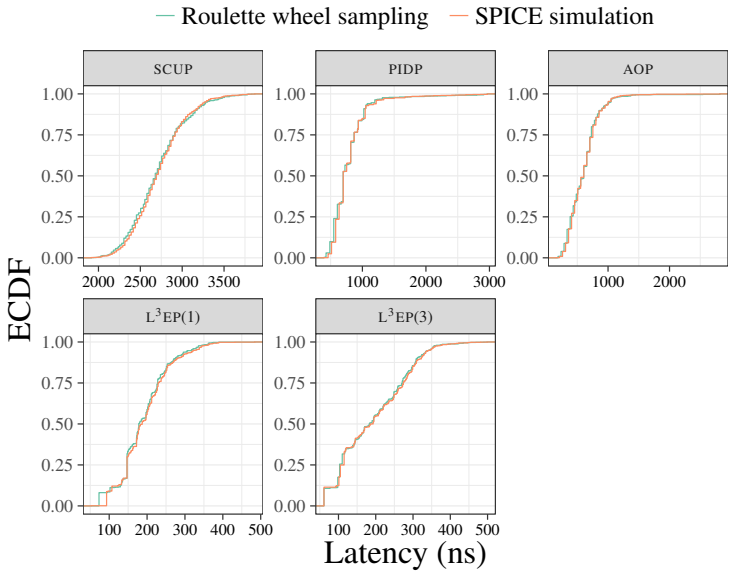

Figure 5.17: The figure compares ECDFs of the latencies as calculated by SPICE and generated by roulette-wheel for the five P&Vs and for $\epsilon = 150\text{K}\Omega$. For brevity, we only show the ECDFs of TLC state 4.

Table 5.4: Workloads used in the full system simulation of the system specified in Table 5.3 [Ham *et al.* 2013].

Workload ID	Benchmarks
WD1	2X leslie3d and 2X mcf
WD2	lbm, leslie3d, libquantum, and mcf
WD3	2X lbm and 2X libquantum
WD4	bwaves, leslie3d, omentpp, and sphinx3
WD5	GemsFDTD, libquantum, milc, and zeusmp
WD6	GemsFDTD, libquantum, and 2X milc
WD7	bzip, libquantum, milc, and omentpp
WD8	cactusADM, gcc, gobmk, and zeusmp
WD9	astar, gobmk, hmmer, and soplex

Table 5.5: L2 cache misses-per-kilo-instruction (MPKI) for the nine workloads of Table 5.4 per P&V per ϵ as computed by gem5 [Binkert *et al.* 2011] and NVMain [Poremba and Xie 2012] using the system parameters as in Table 5.3.

	MPKI									
	$\epsilon = 150\text{K}\Omega$					$\epsilon = 370\text{K}\Omega$				
Workload ID	SCUP	PIDP	AOP	L^3EP(1)	L^3EP(3)	SCUP	PIDP	AOP	L^3EP(1)	L^3EP(3)
WD1	0.38	10.73	11.21	11.08	11.29	11.31	10.97	11.35	11.81	11.75
WD2	13.80	14.09	14.16	14.35	14.27	14.20	14.16	14.28	14.32	14.32
WD3	15.49	15.97	16.01	16.57	16.50	15.98	16.25	16.14	11.46	11.22
WD4	0.37	0.76	0.22	1.88	1.26	0.99	1.05	1.13	2.15	1.98
WD5	0.28	0.72	0.36	1.69	1.64	0.94	1.12	1.17	1.91	1.88
WD6	0.52	0.22	0.23	0.33	0.28	0.23	0.27	0.28	0.39	0.38
WD7	0.33	0.76	0.82	1.24	1.29	0.71	0.93	0.95	1.39	1.38
WD8	0.20	0.42	0.49	0.78	0.78	0.42	0.52	0.56	0.98	0.97
WD9	0.06	0.08	0.09	0.12	0.14	0.09	0.10	0.11	0.04	0.15

Case (i): This case assumes that all the P&Vs have unlimited number of drivers. Fig. 5.18 shows the energies, latencies, bandwidths, and IPCs of every P&V approach averaged across the workloads in Table 5.4 and normalized to SCUP. For $\epsilon = 370\text{K}\Omega$, the latencies of L^{3}EP(1) and L^{3}EP (3) are 2.1–2.8$\times$ lower in comparison to SCUP, PIDP, and AOP. For $\epsilon = 150\text{K}\Omega$, the latencies of L^{3}EP(1) and L^{3}EP(3) are 2.4–8.3$\times$ lower in comparison to SCUP, PIDP, and AOP. As expected, the energy results are similar to the latency results, as energy and latency are directly proportional. For $\epsilon = 370\text{K}\Omega$, the energies of L^{3}EP(1) and L^{3}EP(3) are 1.6–1.9$\times$ lower in comparison to SCUP, PIDP, and AOP. For $\epsilon = 150\text{K}\Omega$, the energies of L^{3}EP(1) and L^{3}EP(3) are 1.6–3.7$\times$ lower in comparison to SCUP, PIDP, and AOP. The IPC results are in agreement with the latency results, i.e., lower latency implies higher IPC. For example, the 90% reduction in latency of L^{3}EP(3) in comparison to SCUP for $\epsilon = 150\text{K}\Omega$ results in 5 $\times$ increase in IPC for L^{3}EP(3) in comparison to SCUP. Whereas PIDP and AOP result in up to 3.33 $\times$ higher IPC than the baseline, L^{3}EP(1) and L^{3}EP(3) result in up to 5 $\times$ higher IPC in comparison to the baseline. For $\epsilon = 370\text{K}\Omega$, the IPCs of L^{3}EP(1) and L^{3}EP(3) are 1.8–2.3$\times$ higher in comparison to SCUP, PIDP, and AOP. Finally, for $\epsilon = 370\text{K}\Omega$, the bandwidths of L^{3}EP(1) and L^{3}EP(3) are 1.8–4.6$\times$ higher in comparison to SCUP, PIDP, and AOP. For $\epsilon = 150\text{K}\Omega$, the bandwidths of L^{3}EP(1) and L^{3}EP(3) are 1.6–2.1$\times$ higher in comparison to SCUP, PIDP, and AOP.

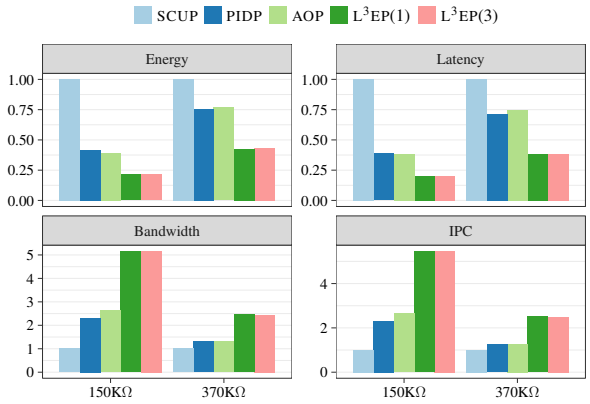

Figure 5.18: This chart shows the energies, latencies, bandwidths, and IPCs of every P&V approach averaged across the workloads in Table 5.4 and normalized to SCUP. This result assumes unlimited write drivers for all the P&Vs evaluated.

Case (ii): This case assumes 16 drivers per memory line for SCUP and 4 drivers per memory line for the rest of the P&Vs evaluated. The results of this case are shown in Fig. 5.19. Clearly, the

results in this case are close to Case (i) result that are shown in Fig. 5.18. This suggests that L^{3}EP (1) and L^{3}EP(3) are at least having $4\times$ better performance than SCUP.

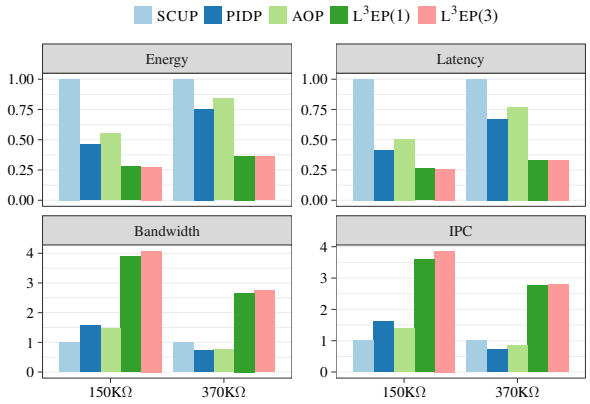

Figure 5.19: This chart shows the energies, latencies, bandwidths, and IPCs of every P&V approach averaged across the workloads in Table 5.4 and normalized to SCUP. This result assumes 16 drivers per memory line for SCUP and 4 drivers per memory line for the rest of the P&Vs evaluated.

Case (iii): This case assumes 16 drivers per memory line for SCUP and 1 driver per memory line for the rest of the P&Vs evaluated. The results of this case are shown in Fig. 5.20. The purpose of this case is to show one particular potential advantage of L^{3}EP over SCUP, i.e., even as the number of drivers in L^{3}EP is $16\times$ less than the number of drivers in SCUP, L^{3}EP still outperforms SCUP with respect to all performance metrics. One observation about Fig. 5.20 is that both PIDP and AOP are having close performance metric to SCUP. In one particular case, the energy consumption of AOP for $\epsilon = 370$ K Ω is higher in comparison to SCUP, and therefore, in this particular case, AOP is used as the baseline energy.

5.2.3 Hardware overhead

The only P&V that has been fabricated in a real chip is SCUP for MLC PCM [Bedeschi *et al.* 2009]. Therefore, it is the baseline for the hardware overhead comparison with L^{3}EP. Both PIDP and AOP require the same hardware overhead as L^{3}EP, and therefore, their hardware overhead is not discussed.

Read path overhead: Both SCUP and L^{3}EP require sense amplifiers for readout operation. L^{3}EP requires an additional circuitry to estimate the resistance of PCM cells. This can be achieved by

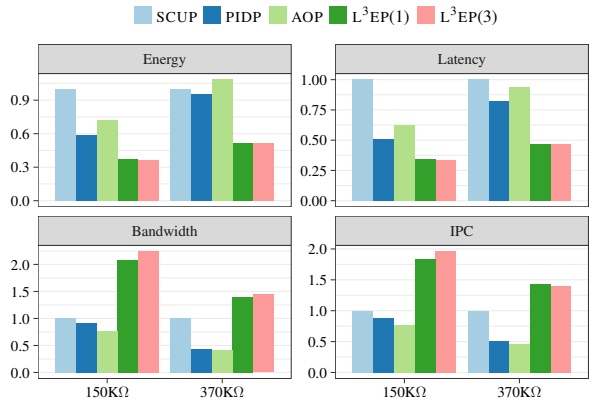

Figure 5.20: This chart shows the energies, latencies, bandwidths, and IPCs of every P&V approach averaged across the workloads in Table 5.4 and normalized to SCUP. This result assumes 16 drivers per memory line for SCUP and 1 driver per memory line for the rest of the P&Vs evaluated.

redirecting the sensed current to an analog-to-digital converter (ADC) that feeds a sample-and-hold (S/H) unit. Therefore, the read path of L^{3}EP requires (i) ADC, (ii) S/H, and (iii) 2–1 MUX to control the current flow from the bitline to either the sense amplifier or resistance estimator. SCUP does not require estimating the resistance. It issues a readout operation after every pulse to check if the target state is reached. On the other hand, readout operations in L^{3}EP are only carried out when a READ command is issued.

Write driver overhead: Every write driver in L^{3}EP requires the following units: a digital-to-analog converter (DAC), an S/H, and a strong voltage follower (VF). These units are connected in series (DAC -> S/H -> VF) and feeds the column decoder as shown in Fig. 5.21. The DAC is controlled by the digital controller which is feeding it with the required voltage of the next pulse. SCUP write driver on the other hand requires RESET and SET charge pumps that consume a major portion of the chip area. In L^{3}EP, these charge pumps are replaced by the DAC, S/H, and strong voltage follower units that are expected to consume comparable chip area. One advantage of SCUP over L^{3}EP is that a single charge pump can be used to program two cells simultaneously. Parallel cell programming in SCUP is possible because all programs (a program is a sequence of pulses) in SCUP have a common start and they differ only in the number of crystallization pulses. In L^{3}EP, even if two cells have the same initial resistance, they may end up having different programs due

to manufacturing variability, and thus, parallel cell programming is not possible in L^{3}EP unless we use more than one independent write drivers.

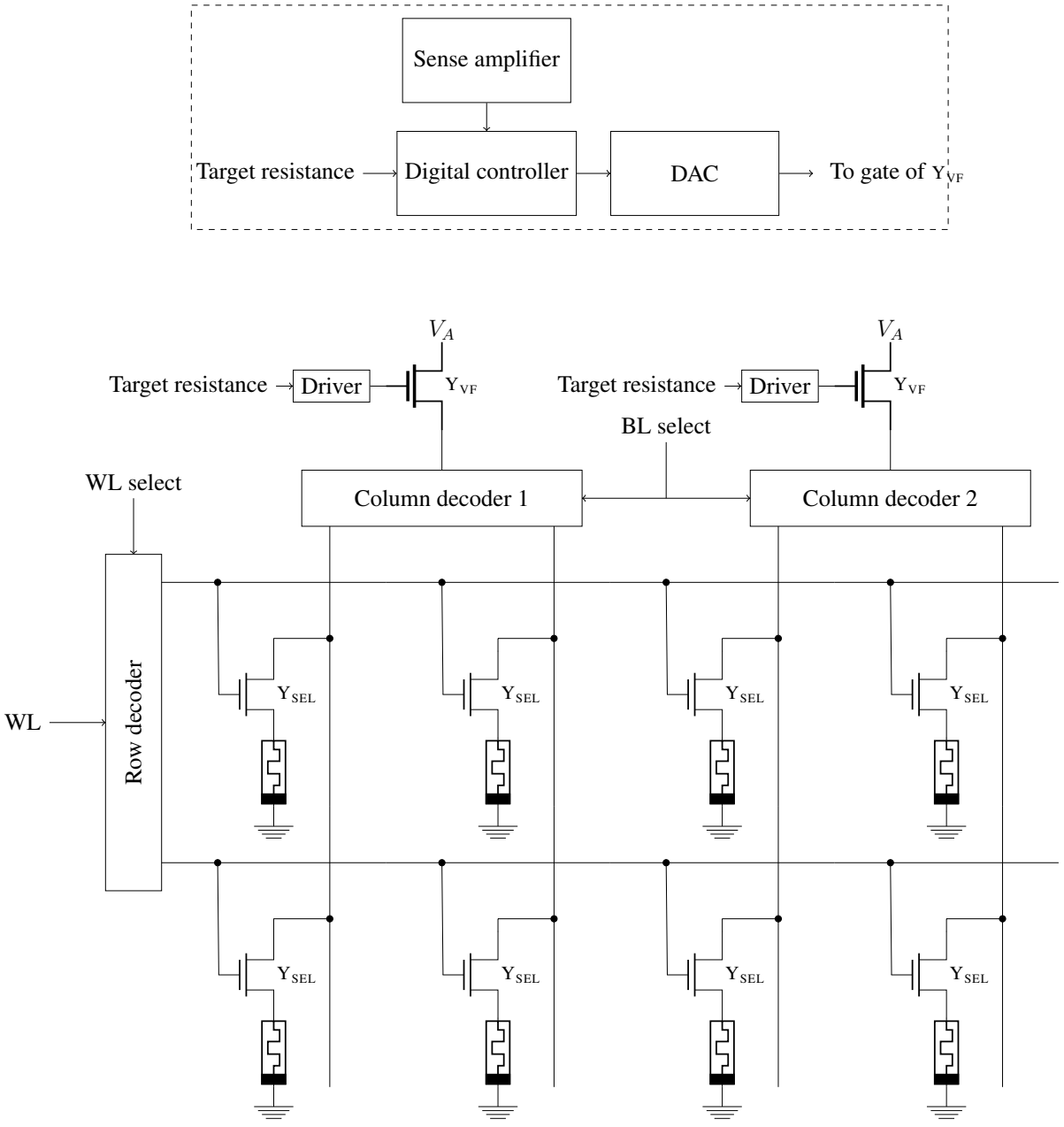

Figure 5.21: This example illustrates how to setup multiple L^{3}EP drivers. The voltage-follower nMOS (VF) is assumed to be strong enough to drive one bit line. The internals of the "driver" block are shown in the subfigure surrounded by a dashed rectangle. It consists of a digital-to-analog converter (DAC), and a sample–and–hold (S/H) circuit. The output of the S/H circuit is connected to the gate terminal of the VF.

We assume L^{3}EP has access to one GRFPU [Catovic 2004], and that the evaluation of the regression model using the linear and cubic predictors takes 4 and 9 cycles, respectively. We also

assume that L^{3}EP needs to track the error, the current and previous value of the pulse amplitude, and the current and previous value of the resistance, resulting in a total of five FP registers per driver that are extra in comparison to SCUP.

5.3 CONCLUSIONS

L^{3}EP is a low latency, low energy P&V approach for TLC PCM based on two core ideas: (i) regression to predict the amplitude of the first amorphization pulse and (ii) packing of crystallization pulses to accelerate crystallization programming. Results show that L^{3}EP is capable of reaching the target TLC state in just one (at most five) pulse (pulses) for 53% (>95%) of the iterations in MC simulations. Furthermore, L^{3}EP reduces the mean P&V latency (energy) by 2.4–15$\times$ (1.9–12.2$\times$) in comparison to state-of-the-art P&V approaches for TLC PCM. L^{3}EP is strongly motivated by the fact that state-of-the-art P&V solutions for PCM have high write energy and latency. Solutions such as L^{3}EP will offset such overheads and accelerate the feasibility and commercialization of MLC/TLC PCM technology.

6.0 FUTURE PLAN

This thesis presents three solutions to mitigate some of the issues of NVMs such as write energy, write latency, and cell endurance. While there are still rooms for improvement in the proposed solutions, there are other active research topics related to NVMs including how to deal with the resistance drift of MLC/TLC PCM [Xu and Zhang 2011, Kim *et al.* 2012, Awasthi *et al.* 2012], how to secure NVMs [Chhabra and Solihin 2011], and how to improve read latency [Hoseinzadeh *et al.* 2014, Yoon *et al.* 2015].

This chapter summarizes two possible directions of future research (i) software–based encoding and (ii) resistance drift technique.

Software–based encoding: The plan is to pursue the integration and evaluation of a software-based hybrid encoding mechanism for MLC/TLC NVMs; this is a page-level encoding that employs a DRAM cache for coding/decoding purposes. The main challenges include how the cache block replacement algorithm can easily access the page-level auxiliary cells to encode the cache block correctly.

Although hardware encoding schemes are usually faster, integrating a software–based encoding scheme enables the realization of complex encoding operations that are otherwise impractical at the hardware level. The primary motivation is to utilize this complexity to achieve more reductions on write energy with lower memory overhead. Complex encoding operations include combining two or more encoding schemes and integrating them into a single encoding scheme. Many combinations of encoding schemes are possible and in the following, we give one example.

CAFO [Maddah *et al.* 2015] has been proposed to reduce the number of bit flips in SLC PCM. Whereas CAFO presents impressive ideas and heuristics to minimize the number of flips, we argue that it is more suitable to be implemented in software due to the complex logic and nested loops it

performs. Note that it might be still possible to implement CAFO in hardware, however, CAFO's logic does not guarantee the number of clock cycles to find the minimal cost code.

Whereas it may be debatable whether CAFO is best implemented in software or hardware, an MLC/TLC version of CAFO is definitely best implemented in software, due to the increased complexity. For example, in MLC CAFO, for every column/row of the data block to be encoded, we need to evaluate four alternative codes of the column/row and pick the one with the highest gain. This needs to be done several times until all column and row gains are ≤ 0. The four codes per row/column can be either generated using MFNW [Alsuwaiyan and Mohanram 2015] or PRES [Seyedzadeh *et al.* 2015] and thus the encoding is hybrid.

Hybrid memory systems utilize NVM as a main memory with a relatively smaller capacity DRAM as a cache, e.g., [Ham *et al.* 2013]. A master memory controller coordinates and manages two disintegrated controllers, one for PCM and another for DRAM. We believe that this architecture can be utilized for to realize the software–based encoding such that the DRAM can serve as a buffer for encoding (decoding) operation before (after) writing (reading) to (from) the NVM.

As indicated above, a major challenge in pursuing this research direction is how to deal with cache block replacement. To illustrate this challenge, let us assume a CAFO/MFNW hybrid encoding scheme. Also, let us assume that a cache block (B) is to be replaced by a new block. Note that if B is dirty, we have to write it back to the NVM; however, the corresponding block in the NVM is encoded at the page level and the auxiliary cells encode the total page. So the questions are:

- How to decode the portion of the page that correspond to the block being replaced?
- How to encode the dirty block B without having to re-encode the whole page?

These as well as any other unforeseen questions need to be addressed before proceeding further in this direction.

Resistance drift: After programming an MLC/TLC PCM cell, the resistance of the cell, $R(t)$, is expected to drift according to the following power law [Burr *et al.* 2010]:

$$R(t) = R_0 \cdot \left(\frac{t}{t_0}\right)^{\nu}$$

Table 6.1: Resistance bounds of some TLC states.

TLC state	Lower bound (MΩ)	Midpoint (MΩ)	Upper bound (MΩ)
5	0.8	1.2	1.6
6	0.02	0.4	0.79

Here, R_0 is the initial resistance after the PCM cell is programmed, t is time, t_0 is the time at which the drift started, and ν is the resistance drift coefficient. The coefficient ν is correlated with R_0 and its value increases as R_0 increases. For the PCM crystalline state, ν is almost zero and that is why drift is minimal in the crystalline state.

Solutions that address the drift problem in MLC/TLC PCM include scrubbing [Awasthi *et al.* 2012] and time–aware sensing [Xu and Zhang 2011]. The scrubbing scheme is similar to DRAM refresh, in which PCM cells are refreshed periodically to prevent them from drifting. In [Awasthi *et al.* 2012], the authors present an affordable error detection scheme that can be embedded within a PCM chip and alerts the memory controller to perform a scrub when an error is detected. The refresh period ranges between 2–512 seconds and is governed by the used error detectiotn schemes. Some schemes provide greater tolerance against uncorrectable errors than others.

In [Xu and Zhang 2011], the authors use information theory to show that using time–unaware ECC, only result in minor reliability improvement. They derive a time–dependent ECC schemes that require require prior knowledge of the lifetimes of the PCM cells.

We propose improving the reliability of ECC schemes for PCM by utlizing the power law in Eq. 6. This law can tell us a lot about R_0. First, we know that $R(t) \geq R_0$. For instance, if $R(t)$ corresponds to TLC state 6 then R_0 can only correspond to TLC state 6 for two reasons (i) TLC state 7 is highly unlikely to drift and (ii) other TLC states have higher resistances than TLC state 6. As another example, if $R(t)$ corresponds to TLC state 5 then R_0 can correspond to TLC state 6 or 5 and we can evaluate which one is more likely by assuming R_0, a suitable ν and t_0, and plug in these values with t (that is given) into Eq. 6 to obtain two values for $R(t)$, one of them is closer to the sensed resistance. We give a numerical example for further illustration.

Example 4: For simplicity, let us assume that t_0 equals 1s. Also, suppose that the bounds of TLC states 5 and 6 are as shown in Table 6.1. Suppose that the resistance $R(t)$ at $t = 10$ s

equals 1.5M Ω, which corresponds to TLC state 5 as per Table 6.1. This resistance could have been drifted from the same TLC state 5 or from TLC state 6. If we assume that it has been drifting from TLC state 5, then we can use ν=0.02 [Xu and Zhang 2011]. We can further assume that $R_0 = 1.2\text{M}\Omega$, i.e., the midpoint resistance of TLC state 5. Substituting these numbers into Eq. 6 gives $R = 1.3\text{M}\Omega$; let us refer to this resistance as R_5, since we assumed it has been drifting from TLC state 5.

On the other hand, if we assume that the resistance has been drifting from TLC state 6 instead, then we can use ν=0.01 [Xu and Zhang 2011]. We can also assume that $R_0 = 0.4\text{M}\Omega$, i.e., the midpoint resistance of TLC state 6. Substituting these numbers into Eq. 6 gives $R = 0.4\text{M}\Omega$; refer to this resistance as R_6.

Recall that we already know the resistance at $t = 10$ s, i.e., $R(10) = 1.5\text{M}\Omega$. By measuring the difference between the actual resistance $R(10)$ and projected resistances R_5 and R_6, we arrive to a conclusion that R_0 is most likely corresponding to TLC state 6, since $|R(10) - R_6| = 0.2\text{M}\Omega \ll |R(10) - R_5| = 1.1\text{M}\Omega$.

The most challenging task in this direction of research is how to validate the idea by statistical reasoning using information theory or by performing Monte Carlo simulations. Note that we need the prediction of R_0 to be accurate within a certain confidence level. In other words, we are aware that this technique can lead to false conclusions sometimes and the questions are as follows:

- How likely are the wrong predictions?
- Can we derive a lower bound on R_0, given R and t? A variation of this question is: Can TLC state 0 drift from TLC state 6? By eliminating an unlikely initial R_0, the calculations become easier and the probability of making the right prediction increases.

These as well as any other unforeseen questions need to be addressed before proceeding further in this direction.

BIBLIOGRAPHY

[Alameldeen and Wood 2004] Alaa R Alameldeen and David A Wood. Frequent pattern compression: A significance-based compression scheme for l2 caches. *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, 1500, 2004.

[Alsuwaiyan and Mohanram 2015] Ali Alsuwaiyan and Kartik Mohanram. MFNW: A flip-n-write architecture for multi-level cell non-volatile memories. In *Proc. Intl. Symposium on Nanoscale Architectures*, 2015.

[Alsuwaiyan and Mohanram 2016] Ali Alsuwaiyan and Kartik Mohanram. An offline frequent value encoding for energy–efficient MLC/TLC non-volatile memories. In *GLSVLSI*, 2016.

[Arjomand *et al.* 2011] M. Arjomand, A. Jadidi, A. Shafiee, and H. Sarbazi-Azad. A morphable phase change memory architecture considering frequent zero values. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, 2011.

[Arnold *et al.* 1992] Jeffrey M Arnold, Duncan A Buell, and Elaine G Davis. Splash 2. In *Proc. Parallel algorithms and architectures*, 1992.

[Awasthi *et al.* 2012] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan. Efficient scrub mechanisms for error-prone emerging memories. In *Proc. Int. Symp. High Performance Computer Architecture*, 2012.

[Baek *et al.* 2004] I.G. Baek, M.S. Lee, S. Seo, M.-J. Lee, D.H. Seo, D.-S. Suh, J.C. Park, S.O. Park, T.I. Kim, I.K. Yoo, U-in Chung, and J.T. Moon. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses. In *Proc. Intl. Electron Devices Meeting*, 2004.

[Bedeschi *et al.* 2009] F. Bedeschi, R. Fackenthal, C. Resta, E.M. Donze, M. Jagasivamani, E.C. Buda, F. Pellizzer, D.W. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE Journal of Solid-State Circuits*, 44(no. 1), 2009.

[Binkert *et al.* 2006] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, Kevin T Lim, Ali G Saidi, and Steven K Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4), 2006.

[Binkert *et al.* 2011] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2), 2011.

[Braga *et al.* 2010] S. Braga, A. Sanasi, A. Cabrini, and G. Torelli. Voltage-driven partial-reset multilevel programming in phase-change memories. *IEEE Trans. on Electron Devices*, 57(10), 2010.

[Burr *et al.* 2010] Geoffrey W Burr, Matthew J Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi, Chung Lam, Luis A Lastras, Alvaro Padilla, et al. Phase change memory technology. *J. Vacuum Science & Technology B*, 28(2), 2010.

[Catovic 2004] Edvin Catovic. *GRFPU-high performance IEEE-754 floating-point unit*, 2004.

[Chang *et al.* 2013] Mu-Tien Chang, P. Rosenfeld, Shih-Lien Lu, and B. Jacob. Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM. In *Proc. Intl. Symposium on High Performance Computer Architecture*, 2013.

[Chatterjee and Hadi 1986] S. Chatterjee and A.S. Hadi. Influential observations, high leverage points, and outliers in linear regression. *Statistical Science*, 1(3), 1986.

[Chhabra and Solihin 2011] Siddhartha Chhabra and Yan Solihin. i–NVMM: a secure non-volatile main memory system with incremental encryption. In *Proc. Int. Symp. Computer Architecture (ISCA)*, 2011.

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

[Choi *et al.* 2012] Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, Junho Shin, Yoohwan Rho, Changsoo Lee, Min Gu Kang, Jaeyun Lee, Yongjin Kwon, Soehee Kim, Jaehwan Kim, Yong-Jun Lee, Qi Wang, Sooho Cha, Sujin Ahn, H. Horii, Jaewook Lee, Kisung Kim, Hansung Joo, Kwangjin Lee, Yeong-Taek Lee, Jeihwan Yoo, and G. Jeong. A 20nm 1.8v 8gb PRAM with 40mb/s program bandwidth. In *Proc. Intl. Solid-State Circuits Conference*, 2012.

[Dgien *et al.* 2014] D.B. Dgien, P.M. Palangappa, N.A. Hunter, Jiayin Li, and K. Mohanram. Compression architecture for bit-write reduction in non-volatile memory technologies. In *Proc. Intl. Symposium Nanoscale Architectures*, 2014.

[Ham *et al.* 2013] Tae Jun Ham, Bharath K Chelepalli, Neng Xue, and Benjamin C Lee. Disintegrated control for energy–efficient and heterogeneous memory systems. In *Proc. Int. Symp. High Performance Computer Architecture*, 2013.

[He *et al.* 2014] Q. He, Z. Li, J.H. Peng, Y.F. Deng, B.J. Zeng, W. Zhou, and X.S. Miao. Continuous controllable amorphization ratio of nanoscale phase change memory cells. *Applied Physics Letters*, 104(22), 2014.

[Hintze and Nelson 1998] Jerry L. Hintze and Ray D. Nelson. Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2), 1998.

[Hoseinzadeh *et al.* 2014] Morteza Hoseinzadeh, Mohammad Arjomand, and Hamid Sarbazi-Azad. Reducing access latency of MLC PCMs through line striping. *ACM SIGARCH Computer Architecture News*, 42(3), 2014.

[ITRS 2011] ITRS. International technology roadmap for semiconductors, 2011 edition. *Semiconductor Industry Association*, 2011.

[Jacobvitz *et al.* 2013] Adam N Jacobvitz, Robert Calderbank, and Daniel J Sorin. Coset coding to extend the lifetime of memory. In *Proc. Int. Symp. High Performance Computer Architecture*, 2013.

[Jiang *et al.* 2012a] Lei Jiang, Youtao Zhang, and Jun Yang. ER: Elastic RESET for low power and long endurance MLC based phase change memory. In *Proc. Intl. Symposium on Low Power Electronics and Design*, 2012.

[Jiang *et al.* 2012b] Lei Jiang, Bo Zhao, Youtao Zhang, Jun Yang, and Bruce R. Childers. Improving write operations in MLC phase change memory. In *Proc. Intl. Symposium on High Performance Computer Architecture*, 2012.

[Kang *et al.* 2008] D-H. Kang, J.-H. Lee, J.H. Kong, D. Ha, J. Yu, C.Y. Um, J.H. Park, F. Yeung, J-H. Kim, W.I. Park, Y.J. Jeon, M.K. Lee, J.H. Park, Y.J. Song, J.H. Oh, H.S. Jeong, and H.S. Jeong. Two-bit cell operation in diode-switch phase change memory cells with 90nm technology. In *Proc. Symposium on VLSI Technology*, 2008.

[Kang *et al.* 2011] M.J. Kang, T.J. Park, Y.W. Kwon, D.H. Ahn, Y.S. Kang, H. Jeong, S.J. Ahn, Y.J. Song, B.C. Kim, S.W. Nam, H.-K. Kang, G.T. Jeong, and C.H. Chung. PRAM cell technology and characterization in 20nm node size. In *Proc. Intl. Electron Devices Meeting*, 2011.

[Kaufman and Rousseeuw 1990] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley, New York, 1990.

[Kim *et al.* 2012] Youngsik Kim, Sungjoo Yoo, and Sunggu Lee. Write performance improvement by hiding R drift latency in phase-change RAM. In *Proc. Design Automation Conference*, 2012.

[Laperle and O'Sullivan 2014] Charles Laperle and Maurice O'Sullivan. Advances in high-speed DACs, ADCs, and DSP for optical coherent transceivers. *J. Lightwave Technology*, 32(4), 2014.

[Lee *et al.* 2008] Kwang-Jin Lee, Beak-Hyung Cho, Woo-Yeong Cho, Sangbeom Kang, Byung-Gil Choi, Hyung-Rok Oh, Chang-Soo Lee, Hye-Jin Kim, Joon min Park, Qi Wang, Mu-Hui Park, Yu-Hwan Ro, Joon-Yong Choi, Ki-Sung Kim, Young-Ran Kim, In-Cheol Shin, Ki won

Lim, Ho-Keun Cho, Chang-Han Choi, Won ryul Chung, Du-Eung Kim, Yong-Jin Yoon, Kwang-Suk Yu, Gi-Tae Jeong, Hong-Sik Jeong, Choong-Keun Kwak, Chang-Hyun Kim, and Kinam Kim. A 90 nm 1.8 v 512 Mb diode-switch PRAM with 266 MB/s read throughput. *IEEE Journal of Solid-State Circuits*, 43(no. 1), 2008.

[Lee *et al.* 2009] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *Proc. Intl. Symposium on Computer Architecture*, 2009.

[Li and Mohanram 2014] Jiayin Li and Kartik Mohanram. Write-once-memory-code phase change memory. In *Proc. Design, Automation and Test in Europe Conference*, 2014.

[Lipowski and Lipowska 2011] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *CoRR*, abs/1109.3627, 2011.

[Liu *et al.* 2012] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-aware intelligent DRAM refresh. In *Computer Architecture News*, 2012.

[Luk *et al.* 2005] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa, and Reddi Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proc. Conference on Programming Language Design and Implementation*, 2005.

[Maddah *et al.* 2015] Rakan Maddah, Seyed Mohammad Seyedzadeh, and Rami Melhem. CAFO: Cost aware flip optimization for asymmetric memories. In *Proc. Int. Symp. High Performance Computer Architecture*, 2015.

[Martin *et al.* 2005] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *ACM SIGARCH Computer Architecture News*, 33(4), 2005.

[Mirhoseini *et al.* 2012] Azalia Mirhoseini, Miodrag Potkonjak, and Farinaz Koushanfar. Coding-based energy minimization for phase change memory. In *Proc. Design Automation Conference*, 2012.

[Mirhoseini *et al.* 2015] A. Mirhoseini, M. Potkonjak, and F. Koushanfar. Phase change memory write cost minimization by data encoding. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(1), 2015.

[Mutlu 2013] Onur Mutlu. Memory scaling: A systems architecture perspective. In *Intl. Memory Workshop (IMW)*, 2013.

[N. and Smith 1981] Draper N. and H. Smith. *Applied Regression Analysis*. 1981.

[Nirschl *et al.* 2007] T. Nirschl, J.B. Philipp, T.D. Happ, G.W. Burr, B. Rajendran, M-H Lee, A. Schrott, M. Yang, M. Breitwisch, C.F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen,

S. Zaidi, S. Raoux, Y.C. Chen, Y. Zhu, R. Bergmann, H. L Lung, and C. Lam. Write strategies for 2 and 4-bit multi-level phase-change memory. In *Proc. Intl. Electron Devices Meeting*, 2007.

[Niu *et al.* 2013] Dimin Niu, Qiaosha Zou, Cong Xu, and Yuan Xie. Low power multi-level-cell resistive memory design with incomplete data mapping. In *Proc. Intl. Conference on Computer Design*, 2013.

[Palangappa and Mohanram 2016] Poovaiah M Palangappa and Kartik Mohanram. CompEx: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVM. In *Proc. Int. Symp. High Performance Computer Architecture*, 2016.

[Papandreou *et al.* 2011] N. Papandreou, H. Pozidis, A. Pantazi, A. Sebastian, M. Breitwisch, C. Lam, and E. Eleftheriou. Programming algorithms for multilevel phase-change memory. In *Proc. Intl. Symposium on Circuits and Systems*, 2011.

[Park and Jun 2009] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2), 2009.

[Pekhimenko *et al.* 2012] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. Base-delta-immediate compression: Practical data compression for on-chip caches. In *Proc. Int. Conf. Parallel Architectures and Compilation Techniques*, 2012.

[Poremba and Xie 2012] Matt Poremba and Yuan Xie. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *ISVLSI*, 2012.

[ptm] Predictive Technology Model, available at http://ptm.asu.edu.

[Schechter *et al.* 2010] Stuart Schechter, Gabriel H. Loh, Karin Strauss, and Doug Burger. Use ECP, not ECC, for hard failures in resistive memories. *ACM SIGARCH Computer Architecture News*, 38(3), 2010.

[Seyedzadeh *et al.* 2015] Seyed Mohammad Seyedzadeh, Rakan Maddah, Alex Jones, and Rami Melhem. PRES: Pseudo-random encoding scheme to increase the bit flip reduction in the memory. In *Proc. Design Automation Conference*, 2015.

[SPEC CPU 2006] SPEC CPU. 2006.

[Stan and Burleson 1995] Mircea R. Stan and Wayne P. Burleson. Bus-invert coding for low-power I/O. *IEEE Trans. on VLSI Systems*, 3(1), 1995.

[Stine *et al.* 2007] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal. FreePDK: An open-source variation-aware design kit. In *Microelectronic Systems Education*, 2007.

[Sun *et al.* 2011] Guangyu Sun, Dimin Niu, Jin Ouyang, and Yuan Xie. A frequent–value based PRAM memory architecture. In *Proc. Asia and South Pacific Design Automation Conference*, 2011.

[Suresh *et al.* 2009] Dinesh Suresh, Banit Agrawal, Jun Yang, and Walid Najjar. Energy-efficient encoding techniques for off-chip data buses. *ACM Trans. on Embedded Computing Systems*, 8(2), 2009.

[Wang *et al.* 2011] Jue Wang, Xiangyu Dong, Guangyu Sun, Dimin Niu, and Yuan Xie. Energy–efficient multi-level cell phase–change memory system with data encoding. In *Proc. Intl. Conference on Computer Design*, 2011.

[Wen *et al.* 2014] Wujie Wen, Yaojun Zhang, Mengjie Mao, and Yiran Chen. State-restrict MLC STT-RAM designs for high-reliable high-performance memory system. In *Proc. Design Automation Conference*, 2014.

[Wong *et al.* 2012] H.-S.P. Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, F.T. Chen, and Ming-Jinn Tsai. Metal–oxide RRAM. *Proceedings of the IEEE*, 100(6), 2012.

[Wu 2012] Junjie Wu. *Cluster Analysis and K-means Clustering: An Introduction*. 2012.

[Xu and Zhang 2011] W. Xu and T. Zhang. A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift. *IEEE Trans. on VLSI Systems*, 19(8), 2011.

[Xu *et al.* 2012] Z. Xu, K. B. Sutaria, C. Yang, C. Chakrabarti, and Y. Cao. Hierarchical modeling of phase change memory for reliable design. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, 2012.

[Xu *et al.* 2013] Cong Xu, D. Niu, N. Muralimanohar, N.P. Jouppi, and Yuan Xie. Understanding the trade-offs in multi-level cell ReRAM memory design. In *Proc. Design Automation Conference*, 2013.

[Yang and Gupta 2002] Jun Yang and Rajiv Gupta. Frequent value locality and its applications. *ACM Trans. Embed. Comput. Syst.*, 1(1), 2002.

[Yang *et al.* 2004] Jun Yang, Rajiv Gupta, and Chuanjun Zhang. Frequent value encoding for low power data buses. *ACM Trans. Des. Autom. Electron. Syst.*, 9(3), 2004.

[Yang *et al.* 2007] Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung gon Yu. A low power phase-change random access memory using a data-comparison write scheme. In *Proc. Intl. Symposium on Circuits and Systems*, 2007.

[Yoon *et al.* 2015] Hanbin Yoon, Justin Meza, Naveen Muralimanohar, Norman P Jouppi, and Onur Mutlu. Efficient data mapping and buffering techniques for multilevel cell phase-change memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(4):40, 2015.

[Yue and Zhu 2012] Jianhui Yue and Yifeng 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.