RICE UNIVERSITY

Coding for Phase Change Memory Performance Optimization

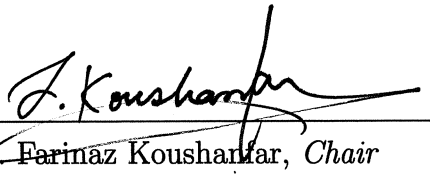
by

Azalia Mirhoseini

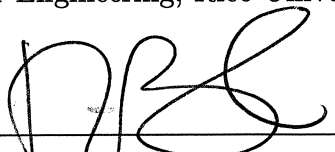
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

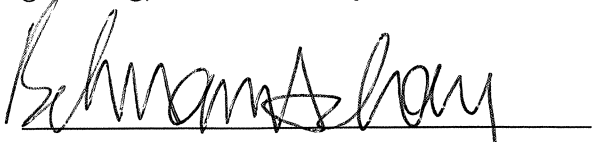
APPROVED, THESIS COMMITTEE:



Dr. Farinaz Koushanfar, *Chair*
Assistant Professor of Electrical and Computer Engineering, Rice University



Dr. Richard Baraniuk
Professor of Electrical and Computer Engineering, Rice University



Dr. Behnaam Aazhang
Professor of Electrical and Computer Engineering, Rice University

HOUSTON, TEXAS
MAY 2012

Abstract

Coding for Phase Change Memory Performance Optimization

by

Azalia Mirhoseini

Over the past several decades, memory technologies have exploited continual scaling of CMOS to drastically improve performance and cost. Unfortunately, charge-based memories become unreliable beyond 20 nm feature sizes. A promising alternative is Phase-Change-Memory (PCM) which leverages scalable resistive thermal mechanisms. To realize PCM's potential, a number of challenges, including the limited wear-endurance and costly writes, need to be addressed. This thesis introduces novel methodologies for encoding data on PCM which exploit asymmetries in read/write performance to minimize memory's wear/energy consumption. First, we map the problem to a distance-based graph clustering problem and prove it is NP-hard. Next, we propose two different approaches: an optimal solution based on Integer-Linear-Programming, and an approximately-optimal solution based on Dynamic-Programming. Our methods target both single-level and multi-level cell PCM and provide further optimizations for stochastically-distributed data. We devise a low overhead hardware architecture for the encoder. Evaluations demonstrate significant performance gains of our framework.

Acknowledgements

The completion of this research would not have been possible without the contribution of several individuals who in one way or another have provided their valuable guidance and assistance.

My deepest gratitude to my advisor Prof. Farinaz Koushanfar for her priceless support, encouragement, and guidance. Her dedication, fondness, and motivation towards doing novel research, as well as exceptional care for her students have taught me invaluable lessons and greatly influenced my life at both academic and personal levels.

Thanks to Prof. Miodrag Potkonjak for sharing his insights and ideas about new directions in optimizing phase-change memory technology.

I would like to acknowledge all the great people who I have been fortunate to work with during my graduate carrier at Rice university for their inspiration and support.

Finally I wish to express profound admiration and gratitude to my beloved parents for their endless love and continuous support for my education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the design of digital integrated circuits, memory often significantly impacts the system's implementation cost, performance, and power dissipation. Presently, there is an ever increasing performance and energy gap between emerging (multi-)processor families and memory. For portable devices and embedded systems with constrained energy sources, minimizing memory energy dissipation is of great importance [1]. Improving and scaling the currently used storage technologies, would have a limited effectiveness in the long run, especially as the miniaturized technologies reach the limits of the silicon. For example, for capacitive memories such as DRAM, scaling beyond 20 nm would likely result in a diminishing capacitor with increased leakage that is unreliable for holding the charge [2]. The newer resistive memory technologies enable an alternative or a hybrid solution that could bridge the growing performance and energy gap between processing and storage.

The data storage mechanism for resistive memories is built upon the large electrical resistance discrepancy between the states of a *phase-change material*. In one phase (state), the material is amorphous and has a very high resistance. In another phase, the same material is crystalline which is highly conductive. Extensive research in the field of phase change storage has demonstrated new material and non-volatile

memory cell structures with improved performance, integration, endurance, retention, and yield properties. Phase-change memory is projected to scale to 9nm [3, 4]. Since several recent work have adopted the PCM terminology for Phase-Change Memory, in the remainder of the manuscript, we use this term.

This thesis aims at minimizing the energy cost of rewriting to the PCM by proposing very low overhead data encoding methods. Our general optimization is easily integrable within the processor architecture and memory interface with a very low complexity and overhead. The method is largely transparent and orthogonal to most other energy saving transformations and methods. Our proposed resistive memory encoding utilizes bitwise manipulation ability during the word overwrites; only the bits that are changing for the new word compared to the existing word in the memory location would require overwriting. Our encoding ensures that the number of required overwrites is minimized. Our optimizations capture asymmetries in energy cost of read, set, and reset operations.

A special case of data encoding for minimizing the unidirectional transitions in the memory is the Write-Once Memory (WOM) coding originally proposed by Rivest and Shamir [5]. They assumed a memory model where the bits could only be set (and could not be reset) and the goal was to increase the number of effective cycles for rewriting to the memory. Subsequent interesting work followed, mostly in information theory and coding with the goal of estimating the capacity and finding more efficient WOM codes. Applications and extension of this model for addressing the flash memory device lifetime improvements were studied [6, 7, 8]. These methods however cannot be directly applied to PCM due to its distinctive energy characteristics. There has been a few work focusing specifically on improving PCM endurance and energy consumption. Reducing the number of cell programming in data updates by avoiding reprogramming redundant bits and flipping the data in case it costs less programming energy has been

suggested [9, 10]. Our work generalizes the above approaches by devising codes for minimizing the energy cost of bi-directional bit transitions for PCM data writes.

The large space of possibilities provided by the freedom in both setting and resetting transitions, and the possibility to perform bit-level operations, motivate the development of new type of codes that can improve the PCM's energy consumption. The complicating factors for this problem are the new degrees of freedom and the curse-of-dimensionality resulting from the exponential number of plausible code combinations. To address the challenge, this thesis presents a novel formal handling of the energy minimization that is appropriate for resistive memory and other storage technologies with bit-level operations and simultaneous consideration of the set and reset transition energy costs. Our contributions are as follows.

- We introduce a formal treatment and formulation of PCM coding, with the goal of minimizing the energy. We show that the problem is NP-complete.
- A methodology for deriving the optimal bounds for minimum-energy data encoding problem is developed.
- We devise a new Integer Linear Programming (ILP) formulation that can find the optimal solution to the problem. Our ILP framework can integrate both symmetric and asymmetric set/reset costs for different code sizes.
- For runtime and efficiency reasons, we develop a new alternative rapid and efficient algorithms for addressing the problem. The method builds upon the smaller optimal codes using Dynamic Programming (DP).
- An efficient distribution-aware data encoding method for non-uniformly distributed data is introduced.
- We discuss and analyze how our method reduce the memory wear.

- An architecture for the coding module is proposed and its overhead is discussed.
- We develop and present a new energy efficient coding for Multi-Level Cell PCM that incorporates its different structure and energy-related properties compared to PCM.
- Evaluation of the proposed encoding methods on a diverse set of data stored on PCM is demonstrated. The data includes a diverse set of benchmark image, audio and text files.

An earlier version of this work appeared in 49th Design Automation Conference (DAC) in San Fransisco, CA [11].

Chapter 2

Related Work and Background

Recent advances in resistive memory material and device technology have paved the way for building PCM devices that are comparable or better than conventional solid state memory and DRAM in terms of certain properties. The field has been rapidly growing in recent years both in research and in terms of industrial prototypes, making PCM the most viable emerging technology for the next generation storage devices [12, 13].

The idea of using the resistance change in phase-change material for storage has been known for more than forty years now [14]. Historically, the performance of resistive memories was not on par with the contemporary solid state and DRAM storage alternatives. During the past 15 years, there has been an unprecedented growth in this technology driven by its desirable characteristics and extensive research in the field. A number of recent work have shown significant improvements in memory performance by integrating PCM within the storage hierarchy [15, 16, 17, 18].

Previous PCM research introduced methods for rewriting to the memory cells such that the writes to all bits have a uniform distribution. The heavily used written lines are remapped to the less frequently utilized locations by the memory management unit [19]. It has been demonstrated that the PCM endurance, reliability, and

energy consumption would greatly improve if the redundant writes are avoided, i.e., by reading the existing contents of the bits and only programming those bits that must be changed. The method is called Data-Comparison-Write (DCW), [9].

Flip-N-Write (FNW) is a protocol that adds an indicator bit to each word to determine if the word is inverted or not, [10]. PCM controller can write the data in an inverted form if it requires less number of bit changes. No optimality proof was provided. Our paper formalizes, provides proofs and generalizes the Flip-N-Write method by devising codes of length $N + K$ for words of length N, where $K \geq 1$. Our approach, for the first time in the literature, considers the asymmetric set and reset energy costs. We will show that significant improvements in energy are achieved over FNW at the expense of allowing a few extra storage bits. We have also devised coding for Multi-Level-Cell PCM.

Some of the existing digital storage mechanisms, including the optical storage, only allow for one directional transition of the bits. Write-Once Memory (WOM) encoding was introduced in a classic paper by Rivest and Shamir [5] to increase the number of writes to such memories with one directional bit setting (in an irreversible fashion). A flurry of subsequent research have centered on improving and generalizing the WOM codes and to extend its reach to other models. The NAND flash memory has been modeled as a one-way transitional memory. Thus, generalizations of the WOM codes have been applied to this class of memories [7, 6, 8].

For the PCM devices, the WOM model and the flash encoding methods do not correctly capture the specifics of the technology. One reason is that the energy discrepancy ratio between the set and reset commands on the PCM is much less subtle when compared to the NAND flash memory devices. The other reason, perhaps even more important, is the ability to perform bit-level manipulation on the PCM, as opposed to block-level operations on NAND flash. The bit-level operations for PCM

have been used earlier for error correcting codes [20]. The work in [20] focused on developing error correction for PCM. Since the faulty bits are rather static, they have demonstrated that Error Correcting Pointers (ECP) that include the knowledge of the fault location, are much more efficient than classic Error Correcting Codes (ECCs). Error correction is orthogonal to our energy efficient data encoding method.

Write-Efficient Memory or WEM is an extension of WOM that has been introduced in [21]. The objective of WEM codes is to minimize the overall number of transitions, and therefore, its goal is close to our encoding case when the set and reset have equal costs. However, to the best of our knowledge, the few papers available on WEM have mainly focused on developing bounds but did not provide an optimality guarantee, or they centered on constructing suitable error correcting codes, e.g., [22, 23]. Aside from the loose bounds and the error correction, we have not been able to find WEM codes that are applicable to the PCM. Besides, we did not find a transform or discussion of the problem's NP-completeness in the earlier literature.

Chapter 3

PCM Operation and Energy Model

3.1 Single-Level PCM

A key challenge for non-volatile memory technology, in particular flash, is the high energy cost of writes [13]. The speed of writing and reading from the caches and from the DRAM is often high, and therefore, the number of transitions is higher than the external memories. Therefore, since resistive memory is suggested for replacing and complementing various storage units in the memory hierarchy, saving the energy cost of set and reset transitions is of a high value [13, 19].

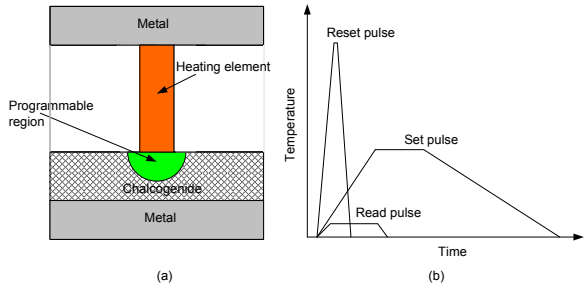


Figure 3.1: (a) The cross section of a conventional PCM memory cell; (b) The flowing current pulses amplitude and duration control the set, reset, and read operations.

As shown in Figure 3.1(a), the current flows through the phase change material(chalcogenide) from the electrode/metal to the heater. This current is provided

as a pulse, and its duration and amplitude controls the temperature needed for the set and reset operations. Heating the phase change material above a crystallization temperature by applying an average current but wide duration pulse results in the set operation. A very high current (melt quenching) pulse with a short duration resets the device to its amorphous state. The read is done by applying a very low amplitude and low power pulse that senses the device resistance. The shape of the three pulses used for set, reset, and read commands is plotted in Figure 3.1(b) (source [13]). The energy discrepancy between the PCM set and reset operations has been experimentally demonstrated and quantified, e.g., [24].

3.2 Multi-Level PCM

The large difference between set and reset resistances in PCM has enabled devising Multi-Level Cell (MLC) PCM. As opposed to the conventional single-level PCM, the randomness and variability in MLC-PCM structure makes it impossible to have a universal pulse shape to attain intermediate resistance levels. Instead, Program and Verify (P&V) is the technique that is used to obtain different resistance distributions for PCM [25]. P&V applies partial program pulses iteratively and then verifies if the desired cell level is achieved. The iterative approach causes MLC PCM to acquire an order of magnitude more write energy than the single level PCM.

One of the main challenges in prototyping MLC PCM is the relaxation effect that induces resistance drift in the phase-change material over the time. The drift is particularly important in MLC-PCM due to the high sensitivity of the cell state level to the resistance value. Different memory sensing and error correction techniques have been proposed to develop more robust MLC PCM systems, [26, 27, 28]. IBM has announced implementing of the first drift-tolerant 2-bit cell PCM in 2011, [29].

Chapter 4

Data Coding Problem

4.1 Coding Overview

Our codes bring energy efficiency by reducing the cost of writes. The efficiency is achieved at the expense of memory overhead; the coded data has a larger length than the actual data. For a given budget of memory overhead, our algorithm develops the codes off-line and its complexity does not affect the realtime performance of the system. The resulting codes from our algorithms are then saved in the memory controller which interfaces to the PCM on one side and to processing units on the other side. Figure 4.1 presents an abstract view of the placement of the data encoding/decoding module for our method. The details of the architecture of the encoder/decoder module and its overhead will be discussed in Chapter 8.

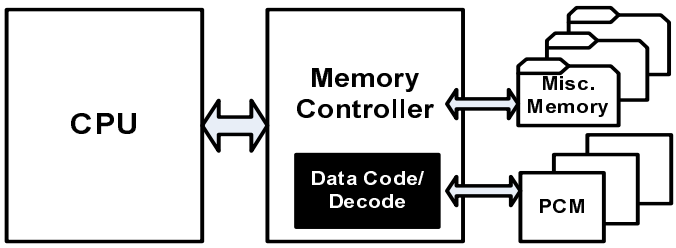


Figure 4.1: Data encoding/decoding module is a part of memory controller in the memory hierarchy.

4.2 Problem Formulation, Complexity and Bounds

Our goal is to minimize the energy cost associated with writing words to the memory with bitwise operability. In this section, each word consists of a fixed number of bits and the energy cost of writing the word is equal to the total cost of the required bit flips, i.e., sets/resets.

We provide an optimal encoding scheme that assigns multiple representations (or codes) to each word in the data set. The objective of encoding is to minimize the energy cost for writing the next word of data. The method trades-off the encoding data overhead with resulting energy improvements. We provide a motivational example to demonstrate the concept more clearly .

4.2.1 A word encoding/decoding Example

In this example, we describe how one may benefit from coding the PCM data. Here we are solving the problem of finding the optimal coding for 2-bit words with 3-bit codes. We denote the words by W_1=(00), W_2=(01), W_3=(10), W_4=(11) and denote the codes corresponding to the word W_i by Z_{i1} and Z_{i2}, for $1 \leq i \leq 4$; since K=1 each word has $2^{K=1}$=2 code representation. The key point is to exploit multiple representations of each word for minimizing the write energy. For instance, if the existing data is Z_{11} and W_2 is to be written on it, among its representations Z_{21} and Z_{22}, the one that incurs the minimum energy cost to overwrite Z_{11} is selected.

Figure 4.2 shows a graph representation of the encodings for the 2-bit words shown in separate clusters. The vertices of the graph are the codes and each cluster represents a word. The graph is a directed graph and the weight of each edge shows the cost of overwriting one node with the other. The optimal encoding is provided

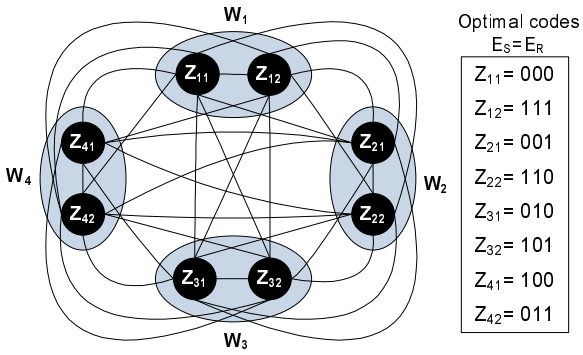


Figure 4.2: A 3-bit encoding for the 4 words W_1, W_2, W_3, and W_4; E_S and E_R are set and reset energies.

on the figure. If the code Z_{22} is to be overwritten by a code of W_3, Z_{31} is selected because its energy cost is only equal to E_S that is the required energy for setting a bit. Denote the bit rest energy by E_R. If no coding was used, overwriting W_2 with W_3 would cost the higher value of E_R+E_S. Another example is a cycle of word overwrites (W_1,W_2,W_3,W_4,W_1). Assume that W_1 is codes as Z_{11}. Then, the minimum cost codes would be selected as follows (Z_{11},Z_{21},Z_{32},Z_{41},Z_{11}). The cost associated with the code overwrites is $E_S+E_S+E_S+E_R+E_R = 2E_S+2.E_R$. Whereas the cost for overwriting the codes without coding is $E_S + (E_S + E_R) + E_S + (2.E_R) = 3.E_S + 3.E_R$.

We have shown that assigning the best codes to each word is equivalent to clustering the vertices of a graph where each cluster represents a word (See Section 4.2.1). Clustering should be done such that it yields the minimum distance between the vertices of different clusters. We can formally define our problem as follows:

Problem. Minimize the energy cost of PCM rewrites.

Given. The word and the codeword (symbol) lengths in bits denoted by N and $N + K$, where $K \geq 1$. Each word is represented by 2^K symbols. The read, set and reset energy are denoted by E_{read} and E_S and E_R respectively.

Objective. Find the best codes for each word so as to minimize the average energy cost of overwrites. We refer to this problem as $\mathcal{P}(N, K)$.

4.2.2 Problem Formulation

We denote the words by $W_1, W_2, \ldots, W_{2^N}$ and denote the codes corresponding to word W_i by Z_{li}, where $1 \leq l \leq 2^K$. Function Φ gives the energy required to overwrite a currently written symbol by a symbol of the next word that would incur the minimum energy cost:

$$\phi(Z_{li}, W'_l) = \min\{C(Z_{li}, Z_{l'i'}), \quad \forall 1 \leq i' \leq 2^K\}. \tag{4.1}$$

The cost function C measures the amount of energy consumed to overwrite a symbol by another one. To overwrite Z_{li} with $Z_{l'i'}$, if N_S number of bit sets and N_R number of bit resets are needed, then C would be:

$$C(Z_{li}, Z_{l'i'}) = (N + K).E_{read} + (N_S).E_S + (N_R).E_R. \tag{4.2}$$

In the above equation, the first term shows the energy for reading the bits of the existing symbol in the memory (Z_{li}). This cost is ignored in our work because of its low value. The next two terms show the energy for the overwrite process (setting and resetting) so as to get $Z_{l'i'}$. Similar bits in the two symbols remain untouched. The Objective Function (OF) can be written as follows:

$$\textbf{OF} : \min\{\mathcal{C}(N, K) = \frac{1}{2^{2N+K}} \sum_{1 \leq l, l' \leq 2^N} \sum_{1 \leq i \leq 2^K} \phi(Z_{li}, W_{l'})\}. \tag{4.3}$$

The challenge is to find the optimal coding of the words that minimizes the OF. Function $\mathcal{C}(N, K)$ represents the average energy cost of code overwrites for all possible rewrites.

4.2.3 Problem Complexity

We have expressed the energy minimizing coding problem as an instance of a distance-based graph clustering problem; each cluster corresponds to a word and the nodes that belong to a cluster are different codes for the cluster's associated word. The goal is to minimize the inter-cluster distances. In the energy minimizing encoding scenario, the inter-cluster distance is the average distance between the code symbols in one cluster and the closest code symbol in every other cluster. Our example demonstrates the interpretation of the data coding as a graph problem (See Section 4.2.1). Extensive prior work on distance-based graph clustering have shown that this problem is NP-hard. The proof was given by a reduction from the set covering problem [30].

4.2.4 Optimal Bounds on the OF

In this part, we provide a lower bound for the OF. The average cost of overwriting each symbol Z_{li} with the other words is determined by the following formulation: $\frac{1}{2^N-1}\sum_{l'}\phi(Z_{li}, W'_l)$ for $l' \neq l$ and $1 \geq l' \leq 2^N$. An optimal code assignment is the one that assigns each of the closest $2^N - 1$ symbols to Z_{li} to one of the words $W'_l \neq W_l$. This assignment gives the minimum average overwrite cost of the symbols.

We provide a lower bound for the OF as follows. First, we calculate the distances from each code Z_{li} to all the other $2^{N+K} - 1$ possible codes. Next, the resulting distances are sorted and the average sum of the smallest 2^N-1 distances are calculated for each node. We compare our DP algorithm result with the optimal bound in our evaluations.

Chapter 5

Solving Energy Efficient Coding Problem

We propose two different approaches for solving the problem formulated earlier. Our first solution is based on mapping the problem to an instance of an Integer Linear Programming (ILP). The method finds the optimal coding for any given word/code width. The approach is discussed in details in the Section 5.1. Due to the complexity of the ILP approach which grows exponentially with the size of the coding problem (i.e., the word and code widths), we introduce another solution based on Dynamic Programming (DP) paradigm. The solution is designed for both uniform and stochastic data and is presented in the Section 5.2.

5.1 Optimal Coding via Integer Linear Programming

An ILP problem formulation requires linear representation of the objective function and the constraints. To the best of our knowledge, ILP has not been used for addressing similar coding problems before. The variables in ILP take integer values. There is a combinatorial complexity associated with assigning values to the variables

of our NP-complete problem. The OF represented in Equation 4.3 is not linear since the function $\phi(.,.)$ is a distance minimization function. To formulate this OF in a linear form, we define variables to indicate the distance of each symbol in a cluster from its closest symbol in every other clusters. The OF is equivalent to the average of all these variables. Certain linear constrains are applied to ensure the variable meets the minimum distance criteria. The ILP method finds the optimal solution at the expense of runtimes exponentially increasing with the code size.

To formulate OF in a linear form, we define an index variable that for each symbol, keeps track of the index of the element (in each of the other clusters) with the minimum distance to the symbol. The following set of variables were used in our ILP formulation:

l, l'	Words indices W_l or W'_l for $1 \leq l, l' \leq 2^N$.
i, i'	Code indices within each cluster, $1 \leq i, i' \leq 2^K$.
Z_{li}	The i-th code $\in W_l$ for all i.
$\Phi_{ll'i}$	$\phi(Z_{l'i}, W_l)$ for all l, l', i and i'.
$w_{ll'ii'}$	$w(Z_{li}, Z_{l'i'})$ for all l, l', i and i'.
$\Delta_{ll'ii'}$	$w_{ll'ii'} - \Phi_{ll'i}$ for all l, l', i and i'.
X_{lij}	j-th significant bit of Z_{li} for $1 \leq j \leq (N+K)$.
$F_{ll'ii'j}$	$w(X_{lij}, X_{l'i'j})$ for all l, l', i, i' and j.
$Id_{ll'ii'}$	An indicator binary; =0 iff $\Delta_{ll'ii'} = 0$ for all l, l', i and i'.

The codes representing a word W_l are shown by Z_{li}; $\Phi_{ll'i}$ denotes the cost of overwriting Z_{li} by a code in $W_{l'}$ that requires the minimum overwrite energy; $w_{ll'ii'}$ is the cost of overwriting two codes U_{li} and $U_{l'i'}$. Thus, $\Phi_{ll'i} = \min_{i'} w_{ll'ii'}$. Each code Z_{li} consists of $N+K$ bits and can be written as $(X_{liN+K}, \ldots, X_{li2}, X_{li1})$. The parameter $F_{ll'ii'j}$ is defined to be the cost of overwriting X_{lij} with $X_{l'i'j}$ and its range of values is shown in the table below. Variable $Id_{ll'ii'}$ is an indicator binary variable

that indicates if the closest code to Z_{il} in cluster l' is $Z_{i'l'}$ or not.

X_{lij}	$X_{l'i'j}$	$F_{ll'ii'j}$
0	0	0
0	1	E_S
1	0	E_R
1	1	0

Using the above variables, we define our OF and provide constraints to our problem in a way that conforms to the ILP format. Our OF, as written in Equation 4.3, minimizes the average cost of overwriting the codes for all possible overwrites:

$$OF : \min \frac{1}{2^N.2^N.2^K} \sum \Phi_{l'li} \qquad \text{for all } l', l \text{ and } i \text{ variables} \tag{5.1}$$

The following constraints define $\Phi_{ll'i}$:

C1. $\Delta_{ll'i} \geq 0$ for all l, l' and i variables,

C2. $\Sigma_{i' \in 1,\ldots,2^k} Id_{ll'ii'} \leq 2^K - 1$,

C3. $Id_{ll'ii'} \leq \Delta_{ll'ii'}$,

C4. $E_R.(N+K).Id_{ll'ii'} \geq \Delta_{ll'ii'}$.

Constraints C1 and C2 set $\Phi_{ll'i}$ not greater than each distance $\Delta_{ll'i}$ and equal to at least one of them respectively; Constraints C3 and C4 define the indicator variable based on the fact that $E_R.(N+K)$ is always grater than $\Delta_{ll'ii'}$.

The below linear constraints set $F_{ll'ii'j}$ to the desired value:

C5. $\frac{1}{E_R+E_S} F_{ll'ii'j} + X_{lij} + X_{l'i'j} \leq 2$,

C7. $F_{ll'ii'j} - E_R.X_{lij} - E_S.X_{l'i'j} \leq 0$,

C8. $F_{ll'ii'j} - E_R.X_{lij} - E_R.X_{l'i'j} \geq 0$,

C9. $F_{ll'ii'j} - E_S.X_{lij} - E_S.X_{l'i'j} \geq 0$.

The following constraint defines the distance $w_{ll'ii'}$:

C10. $w_{ll'ii'} = \Sigma_{1 \leq j \leq N+K} F_{ll'ii'j}$.

The next constraint is set to ensure that no code is assigned to more than one word; E_S is the minimum cost of overwriting two different codes:

C11. $w_{ll'ii'} \geq E_S$.

The output of the above ILP is the values of X_{lij} that constructs the codes U_{il}. The above constraints are all in linear format and can be readily implemented by any ILP solver. The complexity and runtime for solving the instances of the ILP for our NP-complete problem exponentially increases with the instance size. In our experiments, we have been able to find the optimal solution by using a limited version of an ILP solver licensed to one user for N and K ($N = 2, 3, 4,\ K = 1, 2$). If one has access to the commercial ILP solvers that run on the cloud or supercomputers, it is likely possible to find the optimal codes for the practical problems of longer sizes. The longer runtimes can be tolerated since the ILP needs to be used only once and off-line.

5.2 Coding via Dynamic Programming

5.2.1 Coding for Uniform Data

In this subsection, we first show the optimal coding for solving the $\mathcal{P}(N, 1)$. Next, we show how to devise the codes any $\mathcal{P}(N, k)$ based on the coding solutions for the smaller instances of N and K.

Coding For $\mathcal{P}(N, 1)$:

Claim: Optimal coding of $\mathcal{P}(N, 1)$, for any $N \geq 1$ is achieved by assigning the

complement pairs to the words.

Proof: The optimal coding finds $2^K = 2$ symbols, each of size $N+1$, for each word. For now, let us assume that the cost of set and reset is equal. This makes the overwrite cost proportional to the number of bitwise differences for the codes, $E_R = E_S = E$. The average transition cost from each code Z_{li} to all the other words satisfies the following inequality:

$\frac{1}{2^N-1}\sum_{l'}\phi(Z_{li}, W'_l) \;\leq\; 0.\binom{N+1}{0} + E.\binom{N+1}{1} + \cdots +$
$\frac{N-1}{2}E.\binom{N+1}{[\frac{N-1}{2}]} + O.\frac{N+1}{2}E.\binom{N+1}{[\frac{N+1}{2}]}$, *for* $1 \leq l' \leq 2^N$.

Where $O = 1$ if N is odd and $O = 0$ otherwise. The right side of the inequality equals $E.(N+1)2^{N-1}$. The proof of the inequality is as follows. The nearest 2^N codes to Z_{li} should contain all the codes that have zero distance from it (that is Z_{li} itself). The number of such codes is $\binom{N+1}{0}$. It should also include all the codes that are in just one bit different from Z_{li}; the number of such codes is $\binom{N+1}{1}$. The next closest set of codes are the ones that are in different from Z_{li} in 2 bits and so on. We continue until we reach to the first closest 2^N codes to Z_{li}. In that case, the number of bit differences reach to $\frac{N-1}{2}$ when N is even and $\frac{N+1}{2}$ when N is odd. This is because the following equation holds:

$\binom{N+1}{0} + \binom{N+1}{1} + ... + \binom{N+1}{[\frac{N-1}{2}]} + O\binom{N+1}{[\frac{N+1}{2}]} = 2^N$, where O is the same as defined before.

Now, we show that the complement-pair coding assigns all the above 2^N codes to different words. In this case, the average transition cost for each code Z_{li} will be equal to its optimal value and thus the optimal OF is achieved. The sum of bitwise differences of Z_{li} from any complement pair $(Z_{l'1}, Z_{l'2})$, is equal to $N+1$. This is because each bit of Z_{li} is equal to exactly one of the bits of the complement pair. Thus, one symbol of each word has a distance of less than $\frac{N+1}{2}$ bits and the other symbol has a distance of more than $\frac{N+1}{2}$ bits from Z_{li}. This means that all the $2^N - 1$

Algorithm 1. DP-based method for energy-aware coding

Inputs: Word and code lengths: N, N+K; $\mathcal{C}(N,1)$ and optimal coding for $\mathcal{P}(N,1)$ from Section 5.2.1.

⋆ **Finding $\mathcal{C}(n,k)$ and the partitioning index $index(n,k,1:2)$:**

```
1       for (n=1 to n=N)
2           for (k=1 to k=K)
3               if (k==1)
4                   C(n,k) = C(n,1);
5               else
6                   for (i=1 to i=n-1)
7                       for (j=1 to j=k-1)
8                           if (C(n,k) ≥ C(n-i,k-j))
9                               C(n,k) = C(n-i,k-j) + C(ij);
10                              index(N,K,1:2)=(i,j);
```

⋆ **Building the codes for $\mathcal{P}(N,K)$:**

```
11      for (n=1 to n=N)
12          for (k=1 to k=K)
13              if (k==1)
14                  P(n,k) = P(n,1) from Section 5.2.1;
15              else
16                  P(n,k) = all code combinations from P(n - index(n,k,1),
                                                          , k - index(n,k,2))  and
    P(index(n,k,1), index(n,k,2));
```

closest codes to Z_{li} belong to different words.

Note that our complement results for the $K = 1$ case also apply to the asymmetric set/reset costs. The number of sets and resets for traversing from a code to its complement is not symmetric for most of the code words. Recall that our objective is to minimize the average costs over all possible transitions. It can be readily shown that for achieving the mean cost, the average inter-complement distance can replace the two disparate transition costs between the complements. The results of the Lemma 1 then directly follows.

Coding For $\mathcal{P}(N,K)$:

We introduce a DP-based algorithm for solving the general $\mathcal{P}(N, K)$ problem. Our algorithm uses the coding results for $\mathcal{P}(p, q)$ and $\mathcal{P}(r, s)$ to construct the codes for $P(p + r, q + s)$ such that the following bounds can be achieved:

$$\mathcal{C}(p + r, q + s) = \mathcal{C}(p, q) + \mathcal{C}(r, s). \tag{5.2}$$

The code construction is as follows. The word W_i of length $p + r$ is partitioned into 2 words, W_i^1 and W_i^2. The first word is the first p bits and the second word is the last r bits of W_i. There are 2^q, $p + q$-bit symbols for W_i^1 and 2^s, $r + s$-bit symbols for W_i^2 that are obtained from solving $\mathcal{P}(p, q)$ and $\mathcal{P}(r, s)$ respectively. We construct the codes for W_i by concatenating all the possible combinations of these two set of symbols which provides a total of $2^q.2^s = 2^{q+s}$ codes (of length $p + q + r + s$) for W_i. It can be easily seen that the codes satisfy Equation 5.2. Based on the above code construction, the DP method breaks N into smaller values and selects the best partitioning to minimize:

$$\mathcal{C}(N, K) = \min_{i \leq N} \ \{\min_{j \leq i} \mathcal{C}(N - i, K - j) + \mathcal{C}(i, j)\}. \tag{5.3}$$

Algorithm 1 provides the details of the DP method. The optimal coding for $\mathcal{P}(N, 1)$ is given from the previous part and the algorithm iteratively traverses over all the possible partitions to improve the energy minimization objective (Lines 1-10). The index vector $index(n, k, 1 : 2)$ is used to store the optimal partitioning of (n, k). After finding all the indices, the algorithm builds the codes (Lines 11-16). The complexity of the algorithm is $O(N^2K^2)$, but recall that this algorithm is run off-line.

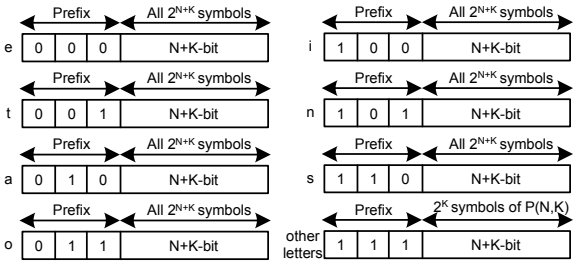


Figure 5.1: Data-aware alphabet letter codings.

5.2.2 Coding For Stochastic Data

In Section 4.2, the OF 4.3 minimizes the average energy cost for all the possible word overwrites. Here, we discuss how the inherent stochastic properties for real data scenarios can be exploited to further improve the memory's energy performance. An important feature is that different words occur with differing frequencies. To benefit from this fact, instead of weighting all the rewrite energy costs equally, we aggressively optimize our encoding for the rewrites that are more prevalent by assigning different number of codes to the words based on their frequency of occurrence.

Variable-length and fixed-length coding are two statistical compression techniques. In the variable-length method, shorter codes are assigned to the more frequent words to better improve the compression. However, this adds to decoding complexity and since our main goal is to minimize the energy, decoding efficiency is very important. Thus, we use a fixed-length coding method. We describe our method on text files that contain English alphabet letters. The method can be generalized to other data sets with nonuniform frequencies. Our data consists of the lower-case alphabet letters: $W_1 = a$, $W_2 = b$, ..., $W_{26} = z$. Since there are 26 letter, W_i's are 5-bit words.

Let us consider the first 7 most frequent letters of the table, e, t, a, o, i, n and s. The probability that an overwrite occurs on any of these letters (by any other letter) plus the probability that these letters overwrite any other letter accounts for almost 60% of all probable overwrites. Thus, we can benefit a lot by optimizing our

coding for these seven letters. To do so, we assign a different prefix to each of these letters such that only the prefixes determine the letter. Since there are 7 letters, the prefixes are 3-bit each and are shown in Figure 5.1. The prefixes can be interpreted as dictionary indices. The remaining $N + K$ bits of these letters take all the possible 2^{N+K} states. Thus, an overwrite to/by any of these letters requires only adjusting the prefix that is of length 3. The other 19 letters have the prefix (111) as shown in the figure. The remaining $N + K$ bits for the less frequent letters are filled with the codes obtained by solving $\mathcal{P}(N, K)$ as described in Subsection 9.2. Thus, an overwrite between the letters costs as much as for a regular $\mathcal{P}(N, K)$. By this coding, we assign 2^{N+K} symbols to the highly frequent letters and 2^K codes to the rest of the letters. All the symbols are of length *prefix-length*$+N + K$.

Chapter 6

Effect of the Encoding on Memory Wear

In this section, we study our encoding scheme in terms of memory wearing. The write endurance of PCM, although orders of magnitude higher than Flash memories, is still limited and considerably less than DRAM. Wear leveling is a technique that is widely used to diminish the limited number of memory write cycles by managing data writes such that they are distributed uniformly across the memory. Wear leveling is performed by memory controller. The encoding scheme can be used along with any conventional wear leveling technique. After the memory controller decides the address to write the data based on the wear leveling method, the encoding module steps in and performs the encoding by first reading the memory at those addresses and then accordingly finding the best codes for the data to be writhen.

The encoding improves the endurance of the memory by reducing the total number of writes (sets and resets). For example, for $\mathcal{P}(2,1)$, the average number of programmings, i.e., average total number of resets and sets, per code write is 0.39 that is equal to 0.13 per bit (since the codes have 2+1=3 bits). However, the average number of programmings per word write if no coding is used is 0.5 that is equal to 0.25 per bit (since the words have 2 bits). In general, the average number of bit flips for rewriting

an N+1-bit code with any other existing code is equal to the following:

$$\frac{1}{N+1}\frac{\Sigma_{1\leq k\leq \lfloor \frac{N+1}{2} \rfloor} k.\binom{N+1}{k}}{2^N} = \frac{(N+1).\Sigma_{1\leq k\leq \lfloor \frac{N+1}{2} \rfloor}\binom{N}{k-1}}{(N+1).2^N} = \frac{1}{2} - \frac{1}{2}\frac{\binom{N}{\lfloor \frac{N}{2} \rfloor}}{2^N.} \tag{6.1}$$

The numerator represents the total number of bit flips required to write an arbitrary code by the closest code (the one that requires less cost) of each of the other 2^N words. As mentioned in the proof of optimal coding for $\mathcal{P}(N,1)$, our coding is designed such that for rewriting a code by the closest code of any other word, the number of bit flips k is in the following range, $1 \leq k \leq \frac{N+1}{2}$. For each k, there are $\binom{N+1}{k}$ of such closest codes, each representing a different word. There is a total of 2^N of such close codes (one for each word) and each code has N+1 bits. Thus, dividing by the denominator yields the average number of flips for each individual bit during a code write. For the N-bit word data, without coding, the average number of bit flips is $\frac{1}{2}$. The proof is straightforward due to the symmetry in the words.

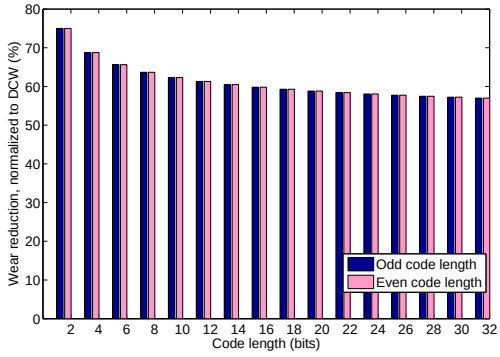


Figure 6.1: This plot shows that PCM write endurance improves with data coding. Each word of length N (bits) is coded by codes of length N+1. The wear efficiencies of 2N-1-bit and 2N-bit codes are equal.

Figure 9.7 shows the improved write endurance with coding compared to DCW method. The words are of length N with the codes of length $N+1$. For example for a 2-bit word with 3-bit codes, the number of allowed writes per bit increases by 50%.

Note that the wear efficiencies of codes of length $2N-1$ and $2N$ are equal. However, the memory overhead of an $2N$-bit code is $\frac{1}{2N}$ which is less than that of an $2N-1$-bit code that is $\frac{1}{2N-1}$. Thus, it is more efficient to use codes with even lengths for saving the memory capacity. Our DP algorithm takes this property into account.

Chapter 7

Multi-Level Cell PCM

The programming energy properties of the MLC-PCM varies from that of a single cell PCM. Thus, we provide our energy encoding optimizations. As mentioned earlier, program and verify is used Table 7.1 shows the average required energies for different levels for a 4-level PCM [25, 31].

Table 7.1: Required energy for programming different levels.

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

Since the required energy for programming the intermediate levels, 01 and 10, is significantly higher than that of 00 and 11, we decided to encode the data such that the number of 01 and 10 cells are minimized. Our method assigns N+1-cell (2N+2-bit) codes to N-cell (2N-bit) data. There are 2^{2N+2} different N+1-cell data. Our coding selects data of length N+1 (Cells) that have the minimum number of intermediate levels and uses them to code the words of length N (Cells). Each N-cell word is coded by one N+1-cell. Here, we provide an example for 2-cell word 3-cell codes in Table 7.2.

Table 7.2: MLC-PCM coding for 2-cell words. The total number of intermediate cells (01 and 10s) in the 2-cell words is 16. The total number of intermediate cells in the corresponding 3-cell codes is 8.

Word	Code
00,00	00,00,00
00,**01**	00,00,11
00,**10**	00,11,00
00,11	00,11,11
01,00	**01**,00,00
01,**01**	**01**,00,11
01,**10**	**01**,11,00
01,11	**01**,11,11
10,00	**10**,00,00
10,**01**	**10**,00,11
10,**10**	**10**,11,00
10,11	**10**,11,11
11,00	11,00,00
11,**01**	11,00,11
11,**10**	11,11,00
11,11	11,11,11

As it can be observed in the table, the total number of intermediate levels for the words is 16, whereas there is only 8 intermediate levels for the corresponding codes. To observe the energy efficiency of the code, we compare the average energy cost for writing uniform data on the memory before and after coding. We denote the four levels 00, 01, 10, and 11 with $L1$, $H1$, $H2$, and $L2$ respectively. For simplicity and due to the symmetry of the problem, we consider the write energies of L1 and L2 to be equal to the average of their individual energies; $\frac{36+20}{2} = 28$. Likewise, $H1$ and $H2$ write energies are considered to be $\frac{307+547}{2} = 427$. For uniform data, where on average all the 16 values are written equal number of times, the average write energy of words is $\frac{8(L1+H1+H2+L2)}{16} = 455$; whereas for coded data, this value equals $\frac{20(L1+L2)+4(H1+H2)}{16} = 183.75$. Thus, on average, the energy is reduced by almost 60%.

In the above example, despite the significant energy saving, the 33% reduction in memory capacity (3-cell codes for 2-cell data) is not desirable. To address this

problem, we propose coding N-cell data with N+1-cell codes for larger N values. In this case, the memory capacity is reduced by a factor of $\frac{1}{N+1}$. The coding uses the same technique as the example and reduces the number of intermediate levels to save energy. We begin with assigning all the codes with zero intermediate levels to the words, then we assign all the codes with one intermediate levels to the words and so on until all the words have a code. Here we calculate how many N+1 cell codes with a given number of intermediate levels, say $0 \leq m$, are available. The answer is equal to the number of N+1-character data with exactly m, $H1$ and $H2$ characters and $N+1-m$, $L1$ and $L2$ characters. From combinatorics, this number is equal to the following.

$$\binom{N+1}{m}.2^{N+1} \qquad \textit{for } 1 \leq m \leq \textit{ N+1.} \tag{7.1}$$

To have the best code for all the N-cell words, we find the minimum m, such that all the 2^{2N} words are covered with codes that has at most m intermediate levels, i.e., the minimum m such that the following inequality holds.

$$\Sigma_{0\leq m}\binom{N+1}{m}.2^{N+1} \leq 2^{2N}. \tag{7.2}$$

We denote the answer by m_{MIN}. Given the answer, the average energy for a code write for the uniform data is $\frac{1}{2^{2N}}\Sigma_{0\leq m\leq m_{MIN}}\binom{N+1}{m}.2^{N+1}.(m\times 427+(N+1-m)\times 28)$. The average write for the corresponding non-coded words is $\frac{\lfloor N\rfloor}{2}(427+28)$; The proof is straightforward due to the symmetry.

Chapter 8

Data Encoder/Decoder Architecture and Overhead

Figure 8.1 shows the architecture of the encoding unit. The read buffer contains the data from the current address and the write buffer contains the new data that is not yet coded. A lookup table is employed for storing the matching codes; given the data in write buffer and the data in the read buffer, the lookup table finds the code that incurs minimum energy for the overwrite. The process of finding the best code from the lookup table causes very low energy overhead due to the low read energy of PCM. Also, since PCM is non-volatile and the leakage power consumption is very small, there is negligible standby power for storing the lookup table [15]. The read latency of PCM is also very low and comparable to that of DRAM [15, 32].

One method to store the lookup table is to store all possible 2^N words and 2^{N+K} codes combinations that means a lookup table of size 2^{N+N+K} . For example, such a lookup table yields to a 2^{15}=32KB table for 8-bit codes (with $N = 7$ and $K = 1$) that is a low memory overhead considering the large memory capacity of today's computer systems ($\geq$GB). However, as the size of the codes grows, the size of the lookup table increases exponentially. To make the lookup table design feasible for large codes, we break the table into sub-lookup tables each containing part of the efficient codes. From our dynamic programming method, each coding problem $\mathcal{P}(N, K)$ is solved by

concatenating the codes from the optimal sub-problem $\mathcal{P}(N_1, K_1)$ and $\mathcal{P}(N_2, K_2)$, where $N_1 + N_2 = N$ and $K_1 + K_2 = K$. For example, $\mathcal{P}(28, 4) = \mathcal{P}(14, 2) + \mathcal{P}(14, 2)$ that can be further break using the following $\mathcal{P}(14, 2) = \mathcal{P}(6, 1) + \mathcal{P}(8, 1)$. Thus, for storing the codes for the original problem, we only need to break the 28-bit word into sub-words of length 6 and 8 and store the codes for the corresponding $2^{6+7=13}$ and $2^{8+9=17}$ sub-words. The result is two lookup tables of sizes 8KB and 128KB.

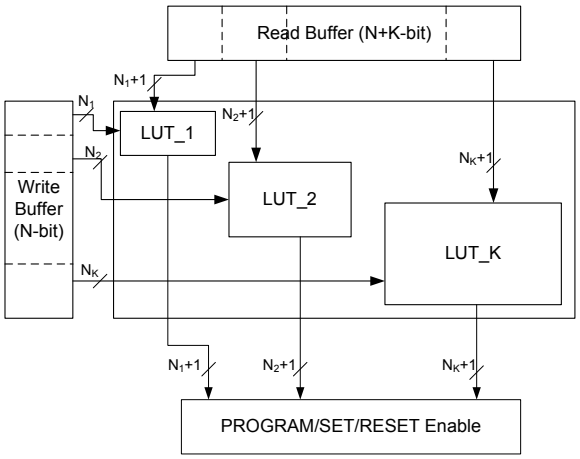


Figure 8.1: This plot shows the architecture of the encoder module. The read buffer contains the old data from PCM and the write buffer contains the new data that is going to be overwritten the read data on PCM.

The Read-Before-Write technique has already been implemented and shown to be promising by designing PCM cache to replace the SRAM cache. The higher density of PCM allows replacing the SRAM cache with a larger capacity cache. Read and write buffers can be used to compensate for the lower speed of PCM compared to SRAM [33].

The main overhead of our coding system will be the memory overhead since in $\mathcal{P}(N, K)$, K extra bits are used to represent an N-bit word, incurring an overhead of $\frac{K}{K+N}$. In our evaluations, we will study the effect of different memory overheads on the system performance.

Chapter 9

Evaluations

We perform system level evaluations of our methods on a variety of real world data sets. The effect of different word widths, different set and reset energies, and the memory and delay overheads on the efficiency of our method is examined.

9.1 ILP Results

We used the latest version of Gurobi ILP solver, Gurobi 4.5.2, to solve the ILP method described in Section 5.1, [34]. Gurobi provides free access for academic purposes. The runtime of the solver for solving $\mathcal{P}(4, 2)$ is about 30 hours on a computer with an Intel dual core 2.80GHz processor and 4GB RAM. Thus, due to the time constraint we were not able to solve the objective function for larger problems. The python ILP code is available upon request to the interested readers.

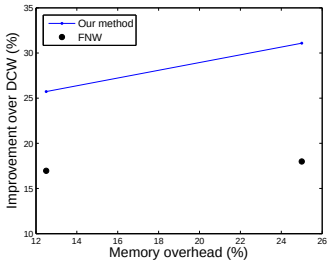


Figure 9.1: 8-bit system

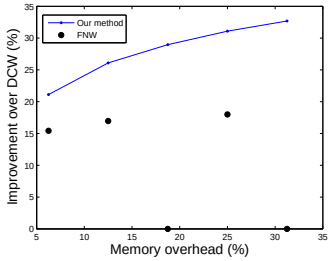


Figure 9.2: 16-bit system

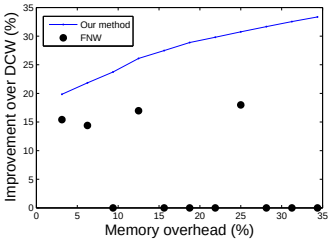


Figure 9.3: 32-bit system

9.2 Performance of DP-based Algorithm on Uniform Data

We analyze DP-based encoding method provided in Algorithm 1 for different memory overheads. We compare our results with Data-Comparison-Write (DCW) and Flip-N-Write (FNW) algorithms [9, 10] described in Section 2.

Here, we show the average efficiencies for uniform data where all the word writes occur with the same frequency. Our metric is the average (per write) energy for all possible word combination overwrites.

Figures 9.1, 9.2, and 9.3 show the energy improvements of our method and the FNW method over the conventional DCW method for 8-bit, 16-bit and 32-bit system respectively. The lined graph shows the results of our method and the black circles show the result of FNW. For example, a 25% memory overhead in a 320bit system means a 32-bit code represents a 24-bit data. The results for FNW system are sparser since they can only take memory overheads of type $\frac{1}{N+1}$. Thus, in a 32-bit system, FNW only accept data-overhead sizes of 31-1, 30-2(15-1), 28-4(7-1), and 24-8(3-1).

It can be seen that our method performs up to 15% better than FNW. There are two main reasons for the better performance. The first reason is the ability of our method to accept different overheads. For example, for solving $\mathcal{P}(30, 2)$ the DP algorithm breaks it into $\mathcal{P}(14, 1) + \mathcal{P}(16, 1)$ as apposed to the FNW approach that

is $2 \times \mathcal{P}(15, 1)$. The former combination, as we discussed in Section 6 delivers better efficiency. Note that as the memory overhead increases, our methods become more efficient. Whereas, in FNW method, $\mathcal{P}(31, 1)$ with memory overhead of 03.13% is slightly more efficient than $\mathcal{P}(30, 2)$ with memory overhead of 6.25%.

The second reason is our focus on energy-efficient selection of the codes to overwrite; FNW always flips the data if the number of bit-flips required to write the original data is more than half of the word's size. However, considering the asymmetric cost of set and reset, we do not count the number of bit-flips; what we count is the total energy of set and resets as our metric to choose a code. For example, let us assume that the new word 00001111 is to overwrite 00000000, FNW writes the new word as it is whereas our method chooses its complement 11110000 to overwrite 000000000. The difference in energy levels is $4 \times E_R$ for FNW versus $4 \times E_S$ in our method, if the ratio of the reset to set energy ($\frac{E_R}{E_S}$) is 2, then our code requires twice less energy.

9.2.1 Performance comparison with respect to the optimal coding

We have compared the performance of our DP-based algorithm with the optimal bound (as presented in Section 4.2.4 and provided the results in the following table. The results shows the average energy cost for all possible data overwrites (see above) with $\frac{E_R}{E_S}$=2. According to the table, the performance gap is increasing with the memory overhead increment, however, the DP algorithm results are still close and in some cases equal to the optimal bound.

Table 9.1: Performance compared to the optimal coding for various code sizes.

Problem size	$\mathcal{P}(4,1)$	$\mathcal{P}(4,2)$	$\mathcal{P}(8,1)$	$\mathcal{P}(8,2)$	$\mathcal{P}(8,3)$	$\mathcal{P}(8,4)$
$\frac{Optimal_{cost}}{DP_{cost}}$	1	1	1	.98	.93	.90

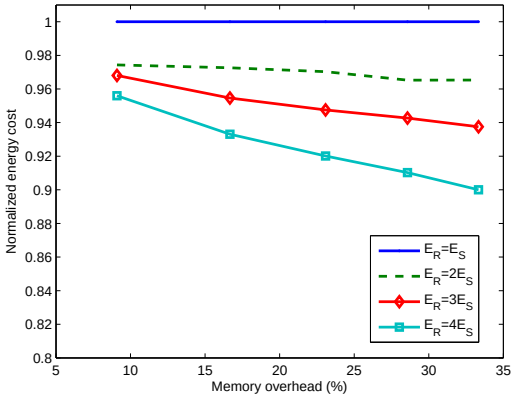


Figure 9.4: Cost reduction by data-aware coding.

9.2.2 Effect of the asymmetric set/reset energy ratios

Here we look at the effect of different $\frac{E_R}{E_S}$ ratios on the efficiency of our method. Figure 9.4 shows the normalized energy costs for various memory overheads. As the ratio increases, more energy savings are achieved; for example, for a memory overhead 30%, if the cost of set and rest is equal the efficiency is % less for $\mathcal{P}(31,1)\frac{E_R}{E_S} = 1$ than for $\mathcal{P}(31,1)\frac{E_R}{E_S} = 4$. This is because our coding scheme aims to optimize the energy consumption by minimizing the number of overwrites. Since resets have a higher energy cost, the minimization impact will be higher for them.

We always consider memory overheads of up to 33.34% (or equivalently Ks up to $\frac{N}{2}$) since no extra efficiency is achieved for $K > \frac{N}{2}$. The reason is that our DP algorithm breaks the $\mathcal{P}(N,K)$ problem for $K > \frac{N}{2}$ to at least one $\mathcal{P}(1,1)$ and it is straightforward to see that coding 1-bit words with 2-bit codes does not provide any efficiency and only incurs memory overhead. Thus, by setting the limit on K we avoid such overheads.

9.3 Performance on Audio and Image Data

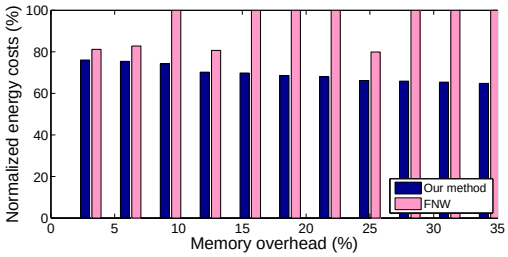


Figure 9.5: Audio data, 32-bit system, $\frac{E_R}{E_S} = 2$.

We use the encoding method for storage of audio and image data on PCM. Our benchmark data were taken from Columbia University audio and Caltech Vision image databases [35] and [36] respectively. Four audio and four image files are selected. The audio data are msmn1.wav, msmv1.wav, mssp1.wav, and msms1.wav and are denoted by $a1$, $a2$, $a3$ and $a4$ in and the image files are $dcp - 2897.jpg$, $dcp - 2898.jpg$, and $dcp - 2899.jpg$ and $dcp - 2830.jpg$.

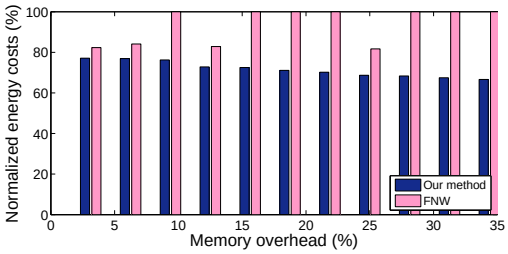


Figure 9.6: Image data, 32-bit system, $\frac{E_R}{E_S} = 2$.

We show the normalized average energy cost of overwriting all the audio files in Figure 9.5 and the image files Figure 9.6. There are 12 possible overwrites for each file type. The costs are shown for 32-bit codings with various memory overheads. PCM holds the following properties PCM: $E_S = 13.733pJ/bit$, and $E_R = 26.808pJ$; the measures are according to for a 32-nm PCM from [17]. The cost of applying our coding method and FNW method are presented. The costs are normalized to the

DCW method's cost. For some memory capacities, FNW cannot be applied and the in such cases its cost is set to be equal to the DCW cost. i.e., 100%.

In both figures, our method outperforms the FNW method. The gap between the performances become wider as the memory overhead increases. Our method outperforms the FNW method by up to 14% and 16%.

9.4 Performance of Stochastic Data Coding

Here we first provide evaluation results for the English alphabet coding as described in Section 5.2.2. Then, we provide coding and evaluations for the ASCII characters. We used two text benchmarks, the 31 MB $text8.txt$ file from [37], for alphabet (excluding spaces) evaluations; and the 4.8 MB $KJV.txt$ file from [38] for ASCII evaluations.

Our evaluations are based on the fact that the overwrites are independent events; for example, the probability that letter a being overwritten by letter b, which we denote by $p(a, b)$ is equal to $p(a).p(b)$, where $p(a)$ and $p(b)$ are the normalized frequencies of the corresponding letters. We experimentally verified the above assumption by randomly selecting 100 vector pairs for overwriting, each of size 100000 from the text benchmark $Text8.txt$. We formed a table of normalized frequencies for all combination of vector pair rewrites and observed that the resulting numbers comply with our independence assumption.

9.4.1 Alphabet Letters

We encoded the alphabet letters with the distribution-aware encoding. Since there are 26 alphabet letters, $N = 5$; we set $K = 1$, and *Prefix*=3. The codes are of length *Prefix*$+N + K = 9$. We evaluated the method on $Text8.txt$ data for different test trials. For each trial, we created 100 pairs of vectors by randomly reading the data

from the text file. Each vector has 1000 letters. We overwrote the vectors of each pair and computed the average overwrite cost for $\frac{E_R}{E_S} = 2$. The results demonstrate an average 44.1% reduction when compared to the no-coding scheme and 9.3% reduction compared to the uniform coding $\mathcal{P}(5, 2)$.

9.4.2 ASCII Characters

According to the frequencies of ASCII characters from [39], 59% of all the possible rewrites are to/by one of the first 15 most frequent characters out of the total 127 characters. Thus, we optimize our coding for these characters by assigning separate prefixes to them.

The first 15 most frequent characters are: space, e, t, a, o, i, n, s, h, r, d, l, u, m, c. We assigned the following 4-it prefixes to them respectively: (0000), (0001), (0010), (0100), (1000), (1001), (1010), (0110), (0111), (1011), (1101). The prefix for all the other characters is (1111). Since there are 2^7 ASCII characters, $N = 7$ and we set $K = 1$. Thus, the codes will be of length $4 + N + K = 12$. The encoding method is the same as described for alphabet letters.

We evaluated the ASCII coding scheme on the *KJV.txt* file. We created 100 pairs of vectors, each of length 1000 from the file. The first vector in each pair was overwritten by the second vector. We considered $\frac{E_R}{E_S} = 2$. To compare this method with the uniform coding, we encoded the ASCII characters with the codes from $\mathcal{P}(7, 1)$, $\mathcal{P}(7, 2)$ and $\mathcal{P}(7, 3)$ and report the corresponding average costs in the following:

Table 9.2: Comparing the ASCII data-aware energy cost with the uniform coding. The costs shows the energy reductions that are normalized to the no-coding method.

Encoding	Data-aware	$\mathcal{P}(7, 1)$	$\mathcal{P}(7, 2)$	$\mathcal{P}(7, 3)$
Average normalized cost (%)	81.6	94.6	91.3	89.3

We see that the ASCII data-aware coding, on average, reduces the energy cost more than the best achieved from $\mathcal{P}(7,3)$; for overwriting each ASCII character, there will be almost 8% more reduction in the energy cost compared to the results of the uniform encoding. This improvement is at the expense of two extra bits per character.

9.5 MLC Coding

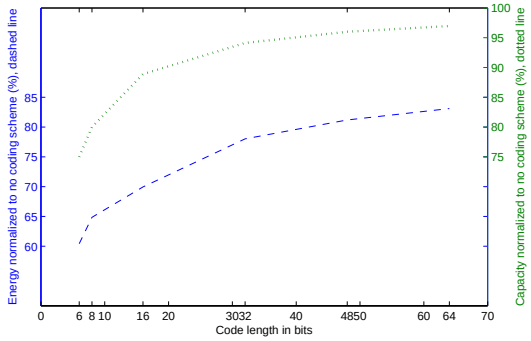


Figure 9.7: This plot shows the results of MLC-PCM proposed coding for energy saving. The dashed line shows the (normalized) average required energy for writing data compared to no-codeing method. The dotted line shows the capacity of the MLC-PCM compared to no-coding method. For example coding a 30-bit (15-cell) word with 32-bit (16-cell) codes results in 0.78 reduction in energy. The capacity is reduced to $\frac{30}{32} = 0.93$ of the full memory capacity.

Figure 9.7 shows the result for the average energy reduction of the coded data for different code widths (N) compared to the non-coded data. Each N-cell word is coded with N+1-cell codes. Thus the overhead for such codes is $\frac{1}{N+1}$. The figure also shows the memory capacity usage of the coded data. It can be seen there is a tradeoff between the energy reduction and the capacity usage. However, the reductions are still significant for a small capacity losses. For example,for 16-cell codes, the write energy is reduced by 78% while 93% of the memory capacity is being used.

Chapter 10

Conclusion

We proposed a novel data coding methodology for minimizing the energy consumption and wear effect of PCM writes. Our method creates several alternative codes for each word on the memory, trading off performance with memory capacity and encoding overhead. The new words to be written on the memory are encoded such that they incur the minimum cost when overwriting the existing words of the memory. To address the coding problem, we developed (i) an ILP-based solution with a high combinational complexity that found the codes optimally; (ii) a Dynamic Programming-based approach that combined the smaller optimal codewords to find near-optimal codes; (iii) and an independent coding approach for Multi-Level Cell PCM that reduced the number of costly intermediate level transitions to improve the performance. For cases where the distributions of the data were a priori known, we created a new data-aware algorithm that incorporated those information for further optimizations. A low overhead architecture for our encoder module was proposed. Evaluations on a diverse set of text, image, and audio benchmark data demonstrated the applicability and effectiveness of our new methods. It was shown that allowing extra memory overhead results in significantly better reductions in memory energy and wear.

References

[1] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," in *International Symposium on Microarchitecture (MICRO)*, 2000, pp. 245–257. 1

[2] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *Micro, IEEE*, vol. 30, no. 1, p. 143, jan.-feb. 2010. 1

[3] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465 –479, july 2008. 1

[4] I. Kim, S. Cho, D. Im, E. Cho, D. Kim, G. Oh, D. Ahn, S. Park, S. Nam, J. Moon, and C. Chung, "High performance pram cell scalable to sub-20nm technology with below 4f2 cell size, extendable to dram applications," in *VLSI Technology (VLSIT), 2010 Symposium on*, june 2010, pp. 203 –204. 1

[5] R. L. Rivest and A. Shamir, "How to reuse a ẅrite - once m̈emory (preliminary version)," in *Symposium on Theory of computing (STOC)*, 1982, pp. 105–113. 1, 2

[6] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Universal rewriting in constrained memories," in *International Symposium on Information Theory (ISIT)*, 2009, pp. 1219–1223. 1, 2

[7] H. Mahdavifar, P. Siegel, A. Vardy, J. Wolf, and E. Yaakobi, "A nearly optimal construction of flash codes," in *International Symposium on Information Theory (ISIT)*, 2009, pp. 1239–1243. 1, 2

[8] Y. Wu and A. Jiang, "Position modulation code for rewriting write-once memories," *IEEE Transactions on Information Theory*, vol. 57, no. 6, pp. 3692–3697, june 2011. 1, 2

[9] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *Circuits and Systems, ISCAS. IEEE International Symposium on*, 2007, pp. 3014 –3017. 1, 2, 9.2

[10] S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," in *International Symposium on Microarchitecture (MICRO)*, 2009, pp. 347–357. 1, 2, 9.2

[11] A. Mirhoseini, M. Potkonjak, and F. Koushanfar, "Coding-based energy minimization for phase change memory," in *Design Automation Conference (DAC)*, 2012, pp. –. 1

[12] S. Lai, "Current status of the phase change memory and its future," in *International Electron Devices Meeting (IEDM)*, 2003, pp. 10.1.1 – 10.1.4. 2

[13] H. Wong, S. Raoux, S. Kim, J. Liang, J. Reifenberg, B. Rajendran, M. Asheghi, and K. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010. 2, 3.1, 3.1

[14] C. Sie, "Memory devices using bistable resistivity in amorphous As-Te-Ge films," PhD dissertation, Proquest/UMI publication 69-20670, Iowa State University, January 1969. 2

[15] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid *PRAM* and *DRAM* main memory system," in *Design Automation Conference*, 2009, pp. 664 –669. 2, 8

[16] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *International Symposium on Computer Architecture (ISCA)*, 2009, pp. 2–13. 2

[17] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *International Symposium on Computer Architecture (ISCA)*, 2009, pp. 14–23. 2, 9.3

[18] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Design exploration of hybrid caches with disparate memory technologies," *ACM Transactions on Architecture and Code Optimization*, vol. 7, December 2010. 2

[19] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *International Symposium on Microarchitecture (MICRO-42)*, 2009, pp. 14–23. 2, 3.1

[20] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *International Symposium on Computer Architecture (ISCA)*, 2010, pp. 141–152. 2

[21] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Information and Computation*, vol. 83, no. 1, pp. 80–97, October 1989. 2

[22] F.-W. Fu and R. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Transactions on Information Theory,*, vol. 46, no. 7, pp. 2299 –2314, November 2000. 2

[23] T. Mittelholzer, L. Lastras-Monta ando, M. Sharma, and M. Franceschini, "Rewritable storage channels with limited number of rewrite iterations," in *International Symposium on Information Theory (ISIT)*, 2010, pp. 973 –977. 2

[24] F. Bedeschi, R. Bez, C. Boffino, E. Bonizzoni, E. Buda, G. Casagrande, L. Costa, M. Ferraro, R. Gastaldi, O. Khouri, F. Ottogalli, F. Pellizzer, A. Pirovano, C. Resta, G. Torelli, and M. Tosi, "4-Mb MOSFET-selected μtrench phase-change memory experimental chip," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1557 – 1565, July 2005. 3.1

[25] F. Bedeschi, R. Fackenthal, C. Resta, E. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. 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," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 1, pp. 217 –227, jan. 2009. 3.2, 7

[26] W. Xu and T. Zhang, "Using time-aware memory sensing to address resistance drift issue in multi-level phase change memory," in *Quality Electronic Design, 2010 11th International Symposium on*, march 2010, pp. 356 –361. 3.2

[27] S. Braga, A. Sanasi, A. Cabrini, and G. Torelli, "Voltage-driven partial-reset multilevel programming in phase-change memories," *Electron Devices, IEEE Transactions on*, vol. 57, no. 10, pp. 2556 –2563, oct. 2010. 3.2

[28] M. Joshi, W. Zhang, and T. Li, "Mercury: A fast and energy-efficient multi-level cell based phase change memory system," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 345 –356. 3.2

[29] N. Papandreou, H. Pozidis, T. Mittelholzer, G. Close, M. Breitwisch, C. Lam, and E. Eleftheriou, "Drift-tolerant multilevel phase-change memory," in *Memory Workshop (IMW), 2011 3rd IEEE International*, may 2011, pp. 1 –4. 3.2

[30] "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, no. 0, pp. 293–306, 1985. 4.2.3

[31] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, oct. 2011, pp. 175 –182. 7

[32] K.-J. Lee, B.-H. Cho, W.-Y. Cho, S. Kang, B.-G. Choi, H.-R. Oh, C.-S. Lee, H.-J. Kim, J.-M. Park, Q. Wang, M.-H. Park, Y.-H. Ro, J.-Y. Choi, K.-S. Kim, Y.-R. Kim, I.-C. Shin, K.-W. Lim, H.-K. Cho, C.-H. Choi, W.-R. Chung, D.-E. Kim, K.-S. Yu, G.-T. Jeong, H.-S. Jeong, C.-K. Kwak, C.-H. Kim, and K. Kim, "A 90nm 1.8v 512mb diode-switch pram with 266mb/s read throughput," in *Solid-State Circuits Conference, ISSCC. Digest of Technical Papers. IEEE International*, 2007, pp. 472 –616. 8

[33] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010, pp. 136–141. 8

[34] "Gurobi ILP solver. http://www.gurobi.com/." 9.1

[35] "Columbia University sound examples directory: http://labrosa.ee.columbia.edu/sounds." 9.3

[36] "Caltech computational vision data repository website: http://www.vision.caltech.edu/html-files/archive.html." 9.3

[37] "Text file test data. http://mattmahoney.net/dc/textdata/." 9.4

[38] "The king james bible (KJV). http://patriot.net/ bmcgin/kjvpage.html." 9.4

[39] "Letter frequency counter. http://millikeys.sourceforge.net/freqanalysis.html." 9.4.2