MFNW: A Flip-N-Write Architecture for Multi-Level Cell Non-Volatile Memories

Ali Alsuwaiyan and Kartik Mohanram
Department of Electrical and Computer Engineering, University of Pittsburgh, PA

Abstract

The increased capacity of multi-level cells (MLC) in emerging non-volatile memory (NVM) technologies comes at the cost of higher cell write energies and lower cell endurance. In this paper, we describe MFNW, a Flip-N-Write encoding solution that effectively reduces the average write energy and improves endurance of MLC NVMs. Two MFNW modes are proposed and analyzed: cell Hamming distance (CHD) mode and energy Hamming distance (EHD) mode. For both modes, we derive a probabilistic model to approximate the statistical behavior of MFNW. For negligible error, the probabilistic model predicts the average number of cell writes per memory word, which is proportional to energy consumption. This enables word length optimization to maximize energy reduction subject to memory overhead constraints. We also estimate the hardware and delay overheads to integrate MFNW into a phase change memory prototype. MFNW is compared to two state-of-the-art techniques in the literature using traces of SPEC2006 benchmarks. Simulation results show an average energy reduction of 19% over the state-of-the-art techniques. Furthermore, we investigate the sensitivity of MFNW to the choice of word length, and our findings suggest a tradeoff between word length and energy reduction.

1. Introduction

The overall performance of a computer system heavily depends on the memory subsystem, especially DRAM. As reported in [1], beyond the 22nm feature size, DRAM scalability slows down due to considerable static and dynamic power requirements. The continuous demand for higher performance systems has led to extensive research to find a more scalable memory technology. Candidates include phase change memory (PCM) [2], spin-transfer torque RAM (STT-RAM) [3], and resistive RAM (RRAM) [4]. These technologies are scalable, have an extended data retention time (i.e., non-volatility), and consume negligible static power. But these NVM technologies also have shortcomings that must be addressed to make them viable replacements for DRAM. For example, programming a memory cell in these NVM technologies requires higher energy than traditional DRAM. Also, NVMs are limited by their endurance : 10^{8-10} writes for PCM and RRAM in comparison to 10^{15} writes for DRAM [2,5,6].

A number of techniques have been proposed to address these challenges in NVMs. Data encoding techniques have been proposed to reduce write energy [7–9]. Wear-leveling techniques that prolong memory lifetime and improve endurance by evenly spreading written data across the memory have been proposed [10–12]. Furthermore, write scheduling and architectural enhancement techniques [13–16] have also been proposed to improve NVM read and write latency. Each one of these techniques usually solves an individual problem (write energy/latency/endurance) of NVM; however, Flip-N-Write (FNW) [17] has the appeal of simultaneously realizing improvements in write energy, endurance, and latency.

FNW, originally introduced as "bus-invert coding" in [18], has been proposed to encode single-level-cell (SLC) memory words in PCM [17]. 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 [19–21]. 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 [22–24].

This paper makes the following contributions. First, we introduce MFNW, a Flip-N-Write algorithm explicitly tailored for MLC NVMs. 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. Third, we develop an approximate probabilistic model to facilitate the theoretical analysis of MFNW. This includes the derivation of a closed form formula for the expected number of cell writes in CHD MFNW. This formula is important because it helps in determining the optimal word length that maximizes write energy reduction subject to memory overhead constraints. Simulation results show that the formula incurs a negligible error of 3% for the chosen sample of word lengths.

Even though MFNW is neutral with respect to the choice of NVM technology, we evaluate it for the two-bits-per-cell MLC PCM prototype proposed in [25]. We estimate the hardware and delay overheads required to implement MFNW for this prototype. We also compare the average write energy of MFNW with two state-of-the-art solutions: data comparison write (DCW) [26] and the data encoding technique in [22]. Note that [27] addresses reducing MLC read energy explicitly, while MFNW reduces write energies. Furthermore, [27] uses only a subset of the codes utilized by MFNW, and therefore, the write energy reduction of MFNW is lower-bounded by the write energy reduction of [27]. Results indicate that MFNW achieves 12–31% energy saving over DCW and 9–22% energy savings over [22]. On average, MFNW results in 19% energy reduction over both techniques. Since the energy reduction of MFNW is sensitive to the choice of the number of cells per word, we also perform sensitivity analysis to illustrate the

This research was supported by King Fahd University of Petroleum and Minerals and by NSF Award CCF-1217738. Corresponding author: kartik.mohanram@gmail.com

effect of word length on energy reduction. Simulation results indicate that as the word length decreases, the energy reductions increase. This suggests a tradeoff between memory overhead (due to tag cells) and energy reduction, as in the SLC case. In summary, for word lengths of 8, 16 and 32 cells, the average energy reductions of MFNW are 23%, 15% and 11%, respectively. Although we did not explicitly evaluate the effect of MFNW on memory endurance, it is widely accepted that reducing write energy also improves NVM endurance [22, 23].

2. Background and Motivation

In this section, we briefly describe how PCM, STT-RAM, and RRAM cells work. Then, we review MLC design in these technologies. DCW is reviewed next, since it serves as the baseline for our evaluations. Finally, we review FNW and how it simultaneously improves write energy, endurance, and latency of SLC NVMs.

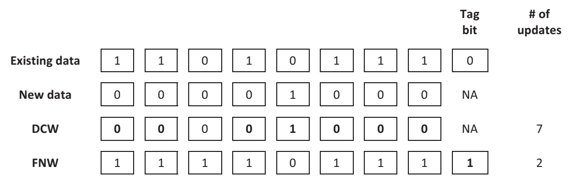
NVM technologies: A PCM cell is made of chalcogenide material which has two states, amorphous and crystalline corresponding with high and low resistances, respectively. To program logic 0 in a PCM cell, current is applied to heat the cell above its melting point followed by a quick cool down. Programming 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 [28, 29].

An STT-RAM cell is composed of two ferromagnetic material layers that sandwich a metal or a metal oxide layer. One of the ferromagnetic layers is called the free layer, since the direction in which its electrons spin can be programmed. The other ferromagnetic layer is fixed with respect to the electrons spin direction. When the electrons in the two ferromagnetic layers spin in parallel, the three layers form a low resistance junction (logic 0). In the anti-parallel case, the layers form a high resistance junction (logic 1). The spin direction of the electrons in the free layer can be programmed by the direction and magnitude of the current passing through the junction [30, 31].

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 [5].

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 [32]. 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) [25] 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.



									Tag bit	# of updates
Existing data	1	1	0	1	0	1	1	1	0	
New data	0	0	0	0	1	0	0	0	NA	
DCW	**0**	**0**	0	**0**	**1**	**0**	**0**	**0**	NA	7
FNW	1	1	1	1	0	1	1	1	**1**	2

Figure 1: **For the shown existing and new data, DCW results in 7 bit-writes. FNW, however, results in only 2 bit-writes.**

Data comparison write (DCW) [26]: 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). 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 [17], 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. 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 [14]. 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.

3. Contributions

This section starts with a high level overview of MFNW, followed by a discussion of the two MFNW modes: CHD and EHD. We also provide a theoretical analysis of CHD MFNW by developing a probabilistic model that helps in the derivation of a closed-form formula for the expected number of cell writes as well as the expected energy consumption. This formula also serves as an upper bound for the expected energy consumption in EHD MFNW.

3.1 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 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. Also, 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.

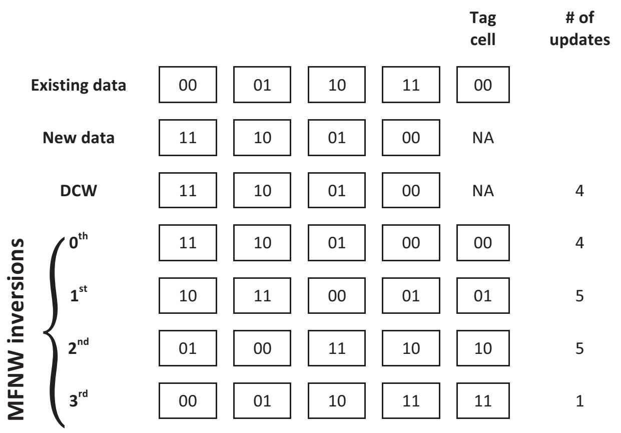
Inversion Operator: Without loss of generality, assume that the MLC is m bits wide, i is a single cell, and a is N cells wide. We define the i^{th} inversion of a as $\{i, \{N\{i\}\} \oplus a\}$. Note the use of both replication and concatenation operators in this definition. Also, note that the i^{th} 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 each cell is 2 bits 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. The algorithm needs to

```
Algorithm 1: MFNW
input  : $W_1$: existing data, $W_2$: tag-less data to write, $q$: 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 $i^{\text{th}}$ inversion of $W_2$, $W_{2,i} = \{i, \{N\{i\}\} \oplus W_2\}$, for
       $0 \leq i < 2^q$.
    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 $A$
```

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: closest with respect to CHD or EHD. We define and illustrate both distance metrics in sub-sections 3.2 and 3.3.

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.



					Tag cell	# of updates
Existing data	00	01	10	11	00	
New data	11	10	01	00	NA	
DCW	11	10	01	00	NA	4
MFNW inversions 0th	11	10	01	00	00	4
1st	10	11	00	01	01	5
2nd	01	00	11	10	10	5
3rd	00	01	10	11	11	1

Figure 2: **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.**

3.2 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. 2 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: '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

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

Note that Y is constrained with a lower bound l. Setting $l = 0$ deactivates this lower bound. We provide the following observation on this lower bound (proof omitted for brevity):

Observation: *Assuming N cells per word, excluding the tag cell, and s states per cell, the number of 'no-cell-write' events is $\geq 1 + \lfloor N/s \rfloor$. Equivalently, the number of 'cell-write' events is $\leq N - \lfloor N/s \rfloor$.*

To determine the probabilities in Eq. 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. 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. 1 is

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

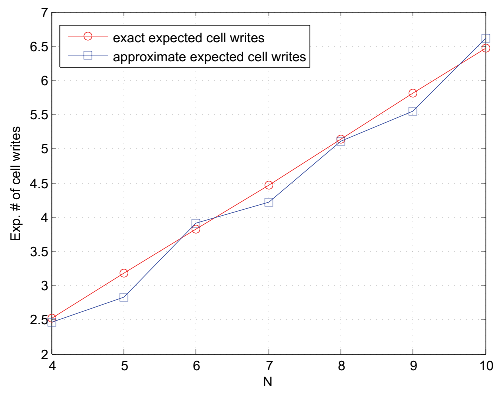


Figure 3: **Expected number of cell writes: numerical computation of exact measure in comparison to the proposed model (Eq. 2.)**

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

Table 1: **Average write energy for MLC prototype [25]**

Substituting these quantities into Eq. 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

$$\overline{W} = N + 1 - \overline{S} \tag{2}$$

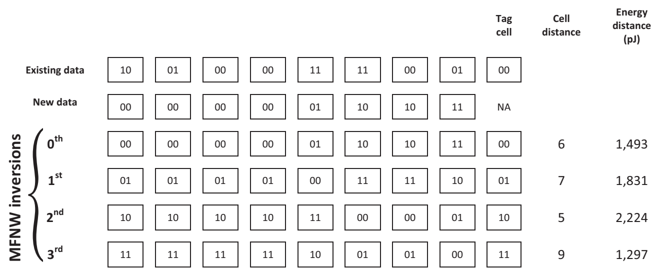
To evaluate the accuracy of this approximation, we computed the average number of cell writes numerically for $N \in \{4, 5, \cdots 10\}$ and compared the result with Eq. 2 in Fig. 3 The geometric mean error is less than 3%, and the error can be as low as 0.5%. It is not surprising to observe that the error for odd N is slightly more than the cases for even N, since the absolute error for the probabilistic model is zero in the SLC case for even N.

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 $1 - \frac{4\overline{W}}{3N}$.

3.3 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 [25]. For this prototype, the energy in our example is 547+307=854 pJ. Table 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. 4 shows a real memory write scenario extracted from memory traces of the 'perlbench' SPEC CPU2006 benchmark [33]. In this example, we observe that the minimum number of cell writes, i.e., 5 writes, occurs



MFNW inversions										Tag cell	Cell distance	Energy distance (pJ)
	Existing data	10	01	00	00	11	11	00	01	00		
	New data	00	00	00	00	01	10	10	11	NA		
0th		00	00	00	00	01	10	10	11	00	6	1,493
1st		01	01	01	01	00	11	11	10	01	7	1,831
2nd		10	10	10	10	11	00	00	01	10	5	2,224
3rd		11	11	11	11	10	01	01	00	11	9	1,297

Figure 4: **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.**

at the 2nd 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 3rd 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 $\overline{W} \times \overline{e}$, where $\overline{W}$ is defined in Eq. 2 and $\overline{e}$ 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.

4. Evaluation and Results

In this section, we present the results of evaluation of EHD MFNW on an MLC NVM architecture. For brevity, we use MFNW to refer to EHD MFNW henceforth. The simulation setup describes the simulators used and their configurations. This is followed by simulation results in which we illustrate the sensitivity of MFNW to word length as a parameter, and compare the energy reduction of MFNW, DCW, and [22]. Finally, we discuss the hardware implementation and delay overheads of MFNW.

Simulation setup: We evaluated MFNW using a trace-driven memory simulator. We extracted the memory traces of a number of floating point and integer SPEC CPU2006 [33] benchmarks using the Pin binary instrumentation tool [34]. During the instrumentation, only main memory access requests were recorded to generate the traces. The trace simulator sorts the trace files by destination address and then by time. In this way, the simulator has $\mathcal{O}(1)$ space-complexity.

Simulation results: We compare average write energies per benchmark in Fig. 5: MFNW and the data encoding technique proposed in [22]. Our chosen word length (excluding the tag cell) is 8 cells (16 bits), and therefore the memory overhead is 1/8=12.5%. Write energies are normalized to DCW. In comparison to DCW, the reduction in write energy for MFNW ranges between 12% and 31%,

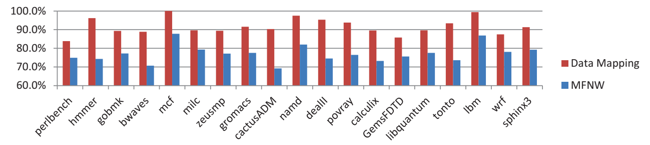


Figure 5: **Energy consumption of DCW and [22] in comparison to EHD MFNW (8 cell/word), normalized to DCW. On average, EHD MFNW results in 19% energy reduction over both techniques.**

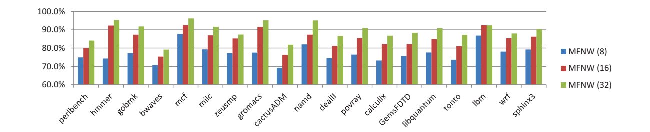


Figure 6: **This chart illustrates MFNW sensitivity to the choice of memory word length. As word length decreases, energy reductions increase, but memory overhead increase. The optimal choice of the word length is subject to energy consumption and space overhead constraints. Data is normalized to DCW.**

with an average of 23%. Note that this is in agreement with the theoretical results, as we reported that lower bound on the expected energy reduction over DCW is 15% for the same size of memory word. In comparison to [22], the reduction in write energy for MFNW ranges between 9% and 22%, with an average reduction of 15%. On average, MFNW has an energy reduction of 19% over both techniques.

Fig. 6 shows how MFNW is sensitive to the choice of word length. We clearly see that the lower the word size, the more the energy reduction. The geometric mean of the energy reductions for 8, 16, and 32 cells word sizes are 23%, 15%, and 11%, respectively. Note again that this agrees with our theoretical findings in which the lower bounds on expected energy reductions for 8, 16, and 32 cells are 20%, 12%, and 9%.

Hardware and delay overheads: We implement MFNW for the MLC PCM prototype proposed in [25]. We also choose the word length to be 8 cells, or 16 bits (excluding the tag cell), since (i) it divides 512, which is our cache line width, 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 simple: bit-wise 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 la-

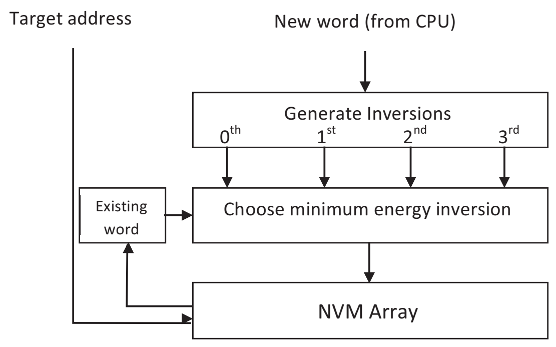


Figure 7: **EHD MFNW write path: The i^{th} inversion is generated as follows: i concatenated with a bit-wise XOR of the new word and $\{8\{i\}\}$. "choose min. energy inversion": counts each state in every inversion and calculates the sum of state count $\times$ state energy for each inversion. The inversion with minimum energy is chosen to replace existing word.**

tency overhead of MFNW read is 0 cycles, since DCW serves as the baseline for our comparisons.

The write path is shown in Fig. 7. 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 "choose minimum energy inversion". 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 "choose minimum energy inversion" 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. 4 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.

Logic overhead, which is roughly 10k gates, is negligible in comparison to memory overhead (12.5%) of tag cells. For the same negligible logic overhead, lower memory overheads (6.25% and 3.125%) are possible, but at the expense of lower energy reductions (lower bounds 12% and 9%, respectively).

5. Conclusions

In this paper, we present an MLC version of SLC FNW, referred to as MFNW that effectively reduces energy consumption and improves endurance of MLC NVM technologies. The key idea that enables MFNW to work effectively is the use of cell inversions, which replaces bit flipping in SLC FNW. We propose and analyze two MFNW modes: CHD MFNW and EHD MFNW. We show that the EHD MFNW leads to more energy reductions. Using a probabilistic model, we derive a closed-form formula for the number of cell writes and energy consumption for both modes of MFNW. This can be used to optimize MFNW word length subject to memory overhead constraints. Energy reduction of MFNW is a function of the word length, and this is demonstrated in the simulation results. Also, we estimate the hardware and delay overheads of EHD MFNW using one of the MLC PCM prototypes. In comparison to two state-of-the-art encoding schemes for MLC PCM, our implementation of EHD MFNW achieves an average of 19% more energy reduction.

6. References

[1] "International Technology Roadmap for Semiconductors," 2011.

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

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

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

[5] H.-S. Wong *et al.*, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, 2012.

[6] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2013.

[7] J. Li and K. Mohanram, "Write-once-memory-code phase change memory," in *Proc. Design, Automation and Test in Europe Conference*, 2014.

[8] D. Dgien *et al.*, "Compression architecture for bit-write reduction in non-volatile memory technologies," in *Proc. Intl. Symposium Nanoscale Architectures*, 2014.

[9] A. Mirhoseini *et al.*, "Coding-based energy minimization for phase change memory," in *Proc. Design Automation Conference*, 2012.

[10] S. Mittal and J. S. Vetter, "EqualChance: Addressing intra-set write variation to increase lifetime of non-volatile caches," *Proc. USENIX Workshop on Interactions of NVM/Flash with OS and Workloads*, 2014.

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

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

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

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

[15] M. K. Qureshi *et al.*, "Scalable high performance main memory system using phase-change memory technology," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[16] ——, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proc. Intl. Symposium on Computer Architecture*, 2012.

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

[18] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. on VLSI Systems*, vol. 3, no. 1, 1995.

[19] T. Nirschl *et al.*, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Proc. Intl. Electron Devices Meeting*, 2007.

[20] D.-H. Kang *et al.*, "Two-bit cell operation in diode-switch phase change memory cells with 90nm technology," in *Proc. Symposium on VLSI Technology*, 2008.

[21] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. Design Automation Conference*, 2013.

[22] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. Intl. Conference on Computer Design*, 2011.

[23] D. Niu *et al.*, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *Proc. Intl. Conference on Computer Design*, 2013.

[24] W. Wen *et al.*, "State-restrict MLC STT-RAM designs for high-reliable high-performance memory system," in *Proc. Design Automation Conference*, 2014.

[25] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE Journal of Solid-State Circuits*, vol. 44, no. no. 1, 2009.

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

[27] H. Hajimiri *et al.*, "Content-aware encoding for improving energy efficiency in multi-level cell resistive random access memory," in *Proc. Intl. Symposium Nanoscale Architectures*, 2013.

[28] K.-J. Lee *et al.*, "A 90 nm 1.8 v 512 Mb diode-switch PRAM with 266 MB/s read throughput," *IEEE Journal of Solid-State Circuits*, vol. 43, no. no. 1, 2008.

[29] M. Kang *et al.*, "PRAM cell technology and characterization in 20nm node size," in *Proc. Intl. Electron Devices Meeting*, 2011.

[30] A. Jog *et al.*, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. Design Automation Conference*, 2012.

[31] W. Xu *et al.*, "Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM)," *IEEE Trans. on VLSI Systems*, vol. 19, no. no. 3, 2011.

[32] M.-T. Chang *et al.*, "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.

[33] "SPEC CPU2006," 2006.

[34] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. Conference on Programming Language Design and Implementation*, 2005.