IEIE Transactions on Smart Processing and Computing, vol. 5, no. 5, October 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.5.337

Bit Flip Reduction Schemes to Improve PCM Lifetime: A Survey

Miseon Han[1] and Youngsun Han[2,*]

[1] School of Electrical and Computer Engineering, Korea University / Seoul, Korea mesunyyam@korea.ac.kr
[2] Department of Electronic Engineering, Kyungil University / Gyeongsan, Korea youngsun@kiu.ac.kr

* Corresponding Author: Youngsun Han

Received August 10, 2016; Accepted September 29, 2016; Published October 30, 2016

** Review Paper: This paper reviews the recent progress possibly including previous works in a particular research topic, and has been accepted by the editorial board through the regular reviewing process.*

***Abstract*:** Recently, as the number of cores in computer systems has increased, the need for larger memory capacity has also increased. Unfortunately, dynamic random access memory (DRAM), popularly used as main memory for decades, now faces a scalability limitation. Phase change memory (PCM) is considered one of the strong alternatives to DRAM due to its advantages, such as high scalability, non-volatility, low idle power, and so on. However, since PCM suffers from short write endurance, direct use of PCM in main memory incurs a significant problem due to its short lifetime. To solve the lifetime limitation, many studies have focused on reducing the number of bit flips per write request. In this paper, we describe the PCM operating principles in detail and explore various bit flip reduction schemes. Also, we compare their performance in terms of bit reduction rate and lifetime improvement.

***Keywords*:** Phase change memory, Bit flip reduction, Data comparison write, Flip-N-Write, FlipMin, CAFO

1. Introduction

As computing environments move to multi-core processors, the demand for larger memory capacity is rapidly increasing. However, further scaling of dynamic random access memory (DRAM) makes the manufacturing process complex, and thus, leads to high manufacturing costs [1, 2]. As a result, DRAM now faces a scalability limitation, even though process technology has improved. Therefore, there is a need for a new type of memory that can be used for main memory instead of DRAM. Among several memory technologies, phase change memory (PCM) has obtained considerable attention as a strong alternative to DRAM due to its advantages, such as high scalability, non-volatility, in-place programmability, low idle power, and reasonable read latency [3, 4].

Uunfortunately, adopting PCM as main memory has a critical problem in that PCM suffers from limited write endurance (about 10^6 to 10^8 writes per cell) [5, 6], which leads to a short lifetime. To overcome the lifetime limitation, many previous works have studied methods such as wear-leveling [7-9], error correction [10-13], bit flip reduction [14-17], and so on [18-20]. Among these solutions, bit flip reduction, which can be orthogonally used with wear-leveling and error correction schemes, is a method that reduces the number of bit flips per write request. This mechanism postpones the wearing out of PCM cells by programming only bits that change their values. In this paper, we explore various bit reduction schemes and evaluate their performance in terms of bit reduction rate and lifetime improvement.

The contributions of this paper can be summarized as follows.

- We explain the details of PCM operating principles.
- We explore various bit flip reduction schemes that decrease the number of bit flips per write operation in order to lengthen PCM lifetime.
- We compare the performance of these schemes in terms of bit flip reduction rate and lifetime improvement.

The rest of this paper is organized as follows. Section 2 describes PCM operations to help understand the background to this paper. Section 3 describes the details of various bit flip reduction schemes. Section 4 explains our evaluation methodology. In Section 5, we analyze the

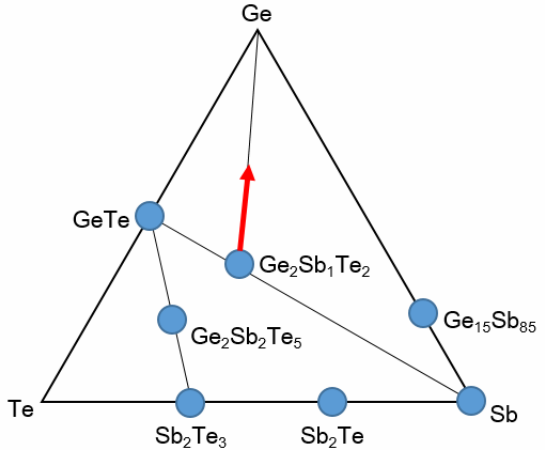


Fig. 1. Tertiary Ge-Sb-Te phase diagram [26].

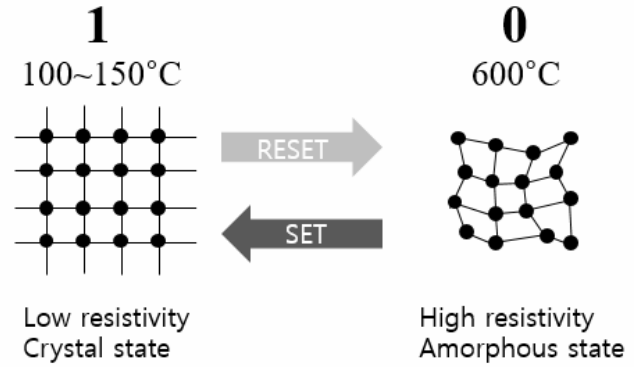


Fig. 2. State change of a PCM cell.

performance of the schemes in terms of bit flip reduction rate and lifetime improvement. Finally, we summarize this paper in the last section.

2. PCM Operating Principles

PCM uses phase change materials in what is called a chalcogenide alloy (GST: $Ge_2Sb_2Te_5$). Fig. 1 shows a ternary phase diagram of the Ge-Sb-Te system. The alloy switches materials along a pseudo-binary line between GeTe and Sb_2Te_3 where compositions can be expressed as $(GeTe)_m(Sb_2Te_3)_n$ [26], e.g., m is 2 and n is 1 for GST. To obtain GST, adding more Ge to the $Ge_2Sb_1Te_2$ starting point leads to the phase change material. To store data in cells, PCM uses a switch between two different physical states of the alloy, i.e., the crystal and amorphous states.

Fig. 2 shows the change in state of a PCM cell. The difference between the two states is a bonding mechanism. The crystal state uses resonance bonding that gives the substance lower potential energy. Alternatively, the amorphous state uses covalent bonding that is stable at a high temperature. Therefore, the crystal state shows low resistivity (1kΩ), whereas the amorphous state has high resistivity (1MΩ). The GST is connected between a bit line and a transistor that is connected to a word line. When the PCM cell is accessed, the word line turns on the transistor, and the datum stored in the transistor is read using

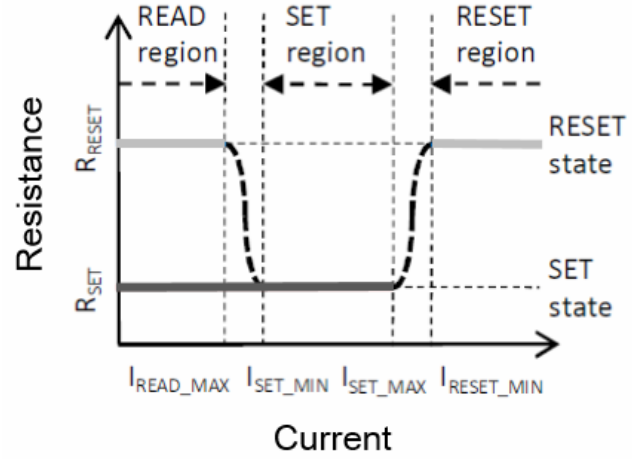


Fig. 3. Typical R-I curve of a PCM cell [15].

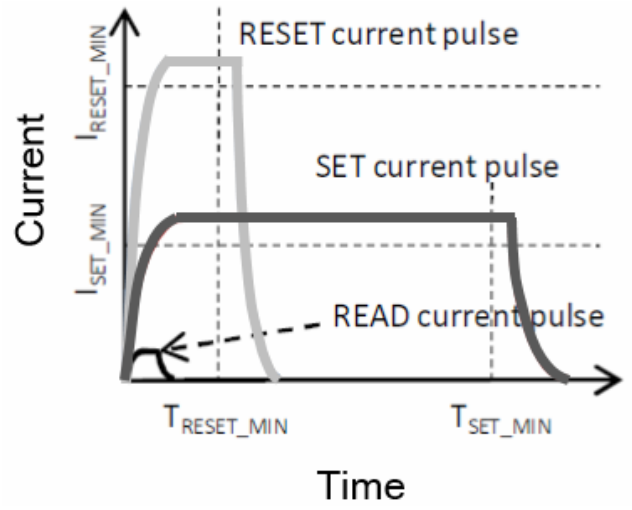


Fig. 4. Current pulses during the read, SET, and RESET operations [15].

resistivity difference.

Fig. 3 shows a typical resistance versus current (R-I) curve of a PCM cell, and Fig. 4 shows the current pulses required to read it, and for SET and RESET. As indicated by the dotted line in Fig. 3, the resistance of the cell is reduced dramatically when the cell hits a certain threshold voltage, V_T. Due to this threshold switching effect, programming the cell is possible. Switching can be done by heating through electrical pulses. The SET current pulse, which has moderate power but long duration, heats GST to around its crystallization temperature (100~150°C), and thus, the cell changes to the SET state. The RESET current pulse, which has high power but a short duration, heats GST above its amorphous temperature (600°C); thus, it puts the memory cell into the RESET state. Since current pulses for SET and RESET operations are intense, a PCM cell gets more unreliable the more it is programmed. As a result, when the number of bit flips exceeds its endurance, it cannot be programmed any more.

```
// A: target address
// D and D': input and previous data
// N: data bit width
1: procedure Write(A, D)
2:   D' = Read(A);
3:   for i=0, i++, i<N do
4:     update a bit if D[i]!=D'[i]
5:   end for
6: end procedure
```

Fig. 5. DCW algorithm.

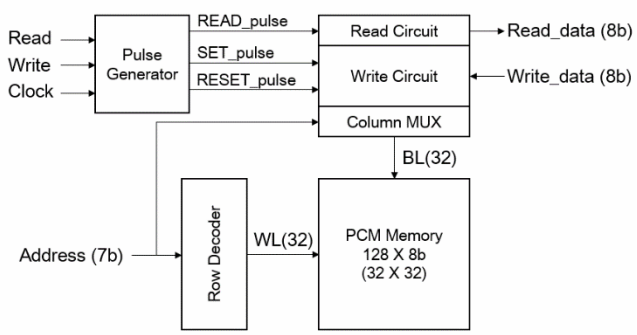


Fig. 6. Simplified block diagram of DCW [14].

3. Bit Flip Reduction Schemes

In this section, we explore several bit flip reduction schemes that reduce the number of bit flips per write request, such as data comparison write (DCW), Flip-N-Write, FlipMin, and cost-aware flip optimization (CAFO).

3.1 DCW

Conventional PCM directly writes input data to a target address, regardless of the previous data stored at the address. If an input datum is 1, it sends a SET current pulse to the target cell. Otherwise, it executes the RESET current pulse on the target cell. Therefore, a bit flip can be 0→0, 0→1, 1→0, or 1→1 in conventional PCM. Among these, bit flips of 0→0 and 1→1 are not necessary, because the previous bit value was the same as the bit value to be stored. Therefore, the DCW scheme skips bit programming in these cases. Fig. 5 shows the DCW algorithm when PCM performs a write request. DCW reads the previous datum from the target address before writing the input datum. If the bit value of the input datum is the same as the current bit value, DCW skips bit programming. Otherwise, DCW programs the cell with the new bit value.

In order to execute the DCW algorithm, the possible implementations are represented in Fig. 6. Row decoder and Column MUX select eight cells with a selected word line and eight selected bit lines by decoding the target address. The pulse generator implements READ_pulse, SET_pulse, and RESET_pulse with Read, Write, and Clock signals. The read circuit and write driver perform the read and write operations using the given pulse and data.

As a result, if each probability of four bit flip cases (0→0, 0→1, 1→0, and 1→1) is evenly 1/4, DCW can halve the number of bit flips, on average, compared to conventional PCM. Also, by reducing bit flips per write operation, DCW can decrease the average power consumption, as shown in Table 1. Since conventional PCM consumes either SET power (P_{SET}) or RESET power (P_{RESET}) regardless of the previous datum when the input datum is 0 or 1, respectively, the average power consumption is $(P_{SET}+P_{RESET})/2$. As a result, DCW can halve the average power consumption by $(P_{SET}+P_{RESET})/4$ just by skipping unnecessary bit programming.

Table 1. Average power consumption of conventional PCM and DCW [15].

Bit flip	Conventional		DCW	
	Power	Probability	Power	Probability
0→0	P_{SET}	1/4	0	1/4
0→1	P_{RESET}	1/4	P_{RESET}	1/4
1→0	P_{SET}	1/4	P_{SET}	1/4
1→1	P_{RESET}	1/4	0	1/4
Average power	$(P_{SET}+P_{RESET})/2$		$(P_{SET}+P_{RESET})/4$	

```
// A: target address
// D and D': input and previous data
// F and F': new and previous flip bit values
// N: data bit width
1 : procedure Write(A, D)
2 :   D' = Read Data(A)
3 :   F' = Read_Flip_Bit(A)
4 :   if hamming_dist({D,0}, {D',F'}) > N/2
5 :     D = ~D
6 :     F = 1
7 :   else
8 :     F = 0
9 :   end if
10:   for i=0, i++, i<N do
11:     update a data bit if D[i]!=D'[i]
12:   end for
13:   update a flip bit if F!=F'
14: end procedure
```

Fig. 7. Write algorithm of Flip-N-Write [15].

```
// A: target address
// D: output data
// F: flip bit value
1 : procedure Read(A)
2 :   D = Read Data(A)
3 :   F = Read Flip Bit(A)
4 :   if F is 1
5 :     D = ~D
6 :   end if
7 : end procedure
```

Fig. 8. Read algorithm of Flip-N-Write [15].

3.2 Flip-N-Write

Even though DCW halves bit flips, on average, if all bit values of input data are different from bit values of previous data, the number of bit flips is N, where N is the data bit widths. Therefore, to further reduce bit flips, Flip-N-Write introduces one additional bit, called a *flip bit*, that indicates whether stored data is inverted or not.

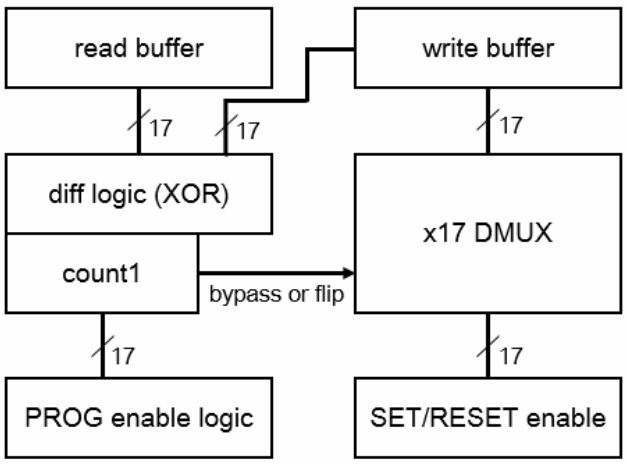


Fig. 9. Logical blocks of the Flip-N-Write data path when data are 16-bit and the *flip bit* is 1 [15].

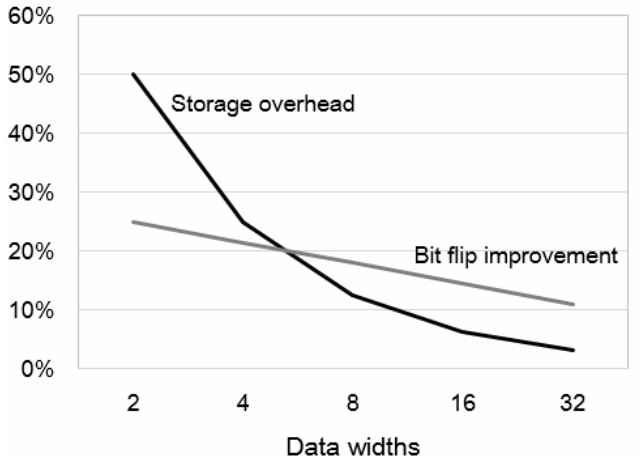


Fig. 10. Improvement with Flip-N-Write against DCW, and storage overhead for Flip-N-Write [15].

```
Dataword1 00 : Coset1 {0000,0101,1010,1111}
Dataword2 01 : Coset2 {0001,0100,1011,1110}
Dataword3 10 : Coset3 {0010,0111,1000,1101}
Dataword4 11 : Coset4 {0011,0110,1001,1100}
```

Fig. 11. Example of FlipMin's coset coding.

Figs. 7 and 8 describe how the Flip-N-Write scheme performs write and read requests, respectively. To perform a write request, Flip-N-Write reads previous data from the target addresses. Next, the Hamming distance of the previous data and *flip bit* from new input data and *flip bit* are calculated to determine the number of bit flips. If the Hamming distance is more than N/2, Flip-N-Write inverts the input data and sets the new *flip bit* to 1. Finally, only bit values of the input data that are different from bit values of the previous data are updated in the target cells. If a new *flip bit* is different from the previous *flip bit*, *flip bit* is also updated. To perform a read request, Flip-N-Write reads previously stored data and the *flip bit*. Then, the data are inverted if the stored *flip bit* is 1.

Fig. 9 shows logical blocks for the Flip-N-Write write data path. Two data from the read buffer and write buffer are compared, bit by bit, using XOR logic, and a bit vector presenting which bits are different is generated. The count1 logic calculates Hamming distances between read and write data that indicate the number of bit flips, to determine whether input data should be flipped or not. The SET/RESET enable signals are generated using data multiplexer output. Also, program-enable signals are generated using XOR logic output and count1 logic determination.

To compare average bit flips in DCW and Flip-N-Write, the average number of bit flips can be calculated with the following equations:

$$F_{DCW} = \sum_{i=0}^{N} i \cdot \frac{1}{2^N} \binom{N}{i} = \frac{N}{2} \quad (1)$$

$$F_{Flip-N-Write} = \sum_{i=0}^{\frac{N}{2}} i \cdot \frac{1}{2^{N+1}} \binom{N+1}{i} + \sum_{i=\frac{N}{2}+1}^{N+1} (N+1-i) \cdot \frac{1}{2^{N+1}} \binom{N+1}{i} \quad (2)$$

where F_{DCW} and $F_{Flip\text{-}N\text{-}Write}$ are average numbers of bit flips in DCW and Flip-N-Write, respectively. In the equations, the probability that both new and old bits are different is assumed to be 1/2. Thus, the probability of having i different bits between new and old data is $(1/2)^N \binom{N}{i}$. As shown in the second equation, the number of bit flips peaks at N/2 and decreases after this point, while DCW keeps increasing the number of bit flips. Fig. 10 plots the bit flip improvement with Flip-N-Write against the DCW scheme and the storage overhead of Flip-N-Write. As shown in the figure, improvement under Flip-N-Write is from 11% (32-bit data) to 25% (2-bit data).

Therefore, Flip-N-Write obtains a further reduced number of bit flips, compared to DCW, by using only an additional *flip bit*. This is because the maximum number of bit flips in Flip-N-Write never exceeds N/2, since it inverts input data and sets a new flip bit value when the Hamming distance is over N/2.

3.3 FlipMin

FlipMin is a bit flip reduction technique that uses coset code [28, 29] when performing write and read requests. It encodes an input dataword into a vector that is actually to be stored in PCM cells while performing a write request. Also, it decodes a stored vector to an output dataword during a read request. Coset coding provides many vectors to the same dataword with extra bits. In case of k-bit dataword and n-bit vector with c-bit additional bits, i.e., $c = n - k$, a dataword and vector can be one of 2^k and 2^n strings of zeros and ones, respectively. This means one dataword of 2^k can have a set with 2^c vectors. FlipMin chooses only one vector among a set that is called a coset while encoding the dataword to minimize the number of bits.

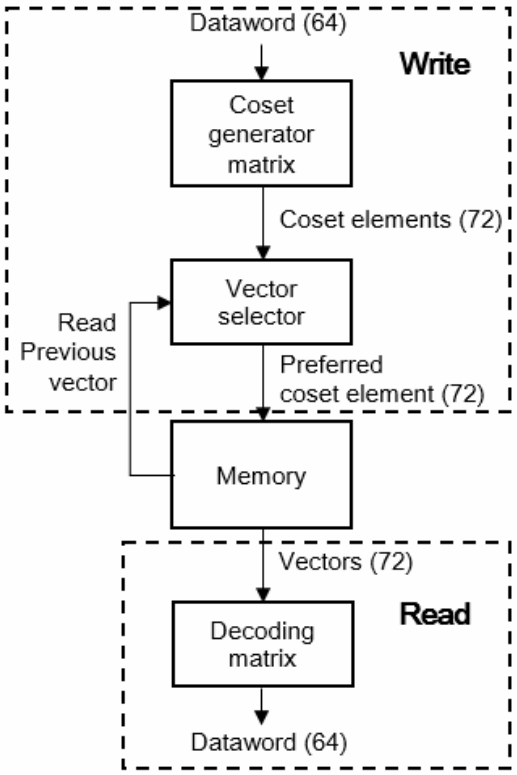


Fig. 12. Logical blocks of FlipMin [16].

Fig. 11 presents an example of FlipMin's coset coding when datawords are two-bit and vectors are four-bit, i.e., extra bits are two-bit. FlipMin maps each dataword to a coset that consists of four vectors. When PCM wants to write data to cells, if the dataword to be written is 01 and the previously stored vector is 1001, FlipMin maps a dataword 01 to coset2, and chooses the vector 1011 that has the minimum bit flips, and stores it. When PCM wants to read data from the cells, if the stored vector at the target address is 0111, the vector is decoded into dataword 01.

To generate cosets, FlipMin uses dual Hamming (72,64) code, called punctured Hadamard code [27], that maps 8-bit data to 72-bit vectors with 64-bit parities. In the code, there are many vectors that have the same parity, i.e., 2^8 vectors for one parity. Therefore, FlipMin considers a 64-bit dataword as parity code and 72-bit vectors that have the same parity as cosets. When FlipMin writes a dataword to PCM cells, it maps the dataword to parity code and finds the vector with the minimum bit flips from a coset that has the same parity code.

Fig. 12 shows the logical blocks for writing/reading in FlipMin. When PCM writes a dataword, the coset generator matrix maps the dataword to a set and generates coset elements. Then, PCM reads the previous vector from memory, and selects from coset elements the vector that has the minimum bit flips. This selected vector is written to memory. In contrast, when PCM reads a dataword, the vector read from memory is converted to the dataword through the decoding matrix.

With coset coding, FlipMin can reduce bit flips further than both DCW and Flip-N-Write. Table 2 shows the average number of bit flips in FlipMin when datawords are four-bit. With DCW, the number of datawords that have no 1's is one, and that have one 1 is four, and so on. Therefore, the average number of bit flips in DCW is $(0\times1+1\times4+2\times6+3\times4+4\times1)/16=2$. With FlipMin, the average number of bit flips is 1.375, i.e., $(0\times1+1\times8+2\times7+3\times0+4\times0)/16=1.375$, which is a 31% reduction. For 64-bit datawords, the average number of bit flips in DCW, Flip-N-Write, and FlipMin are 32, 28.82, and 24.48, respectively. Therefore, FlipMin reduces average bit flips by 24.5%, compared to DCW, which is significantly larger than the 9.93% in Flip-N-Write.

Table 2. Average number of bit flips when datawords are four-bit [16].

DCW		FlipMin	
# of 1's	# of words	# of 1's	# of words
0	1	0	1
1	4	1	8
2	6	2	7
3	4	3	0
4	1	4	0
Bit flips	2.0	Bit flips	1.375

3.4 CAFO

Although DCW, Flip-N-Write, and FlipMin significantly reduce the number of bit flips, they do not consider the asymmetric nature of a PCM cell's state change. Since RESET current pulse is more intense than the SET current pulse, PCM cell endurance is more dependent on the RESET current [24, 25]. Therefore, CAFO proposes a cost model considering the asymmetric nature of programming a PCM cell, and uses it to encode and decode data to reduce the overall cost of performing a write request.

When comparing previous and input data, there are four different bit flip cases. CAFO labels the cost of these bit flip cases, which are 0→1, 1→0, 0→0, and 1→1, to *a*, *b*, *c*, and *d*, respectively and models the total cost of a write request, named *C,* with the following formula:

$$C = n_{0\to1}a + n_{1\to0}b + n_{0\to0}c + n_{1\to1}d \tag{3}$$

where $n_{0\to1}$, $n_{1\to0}$, $n_{0\to0}$, and $n_{1\to1}$ are the number of *a*, *b*, *c*, and *d* bit flip cases, respectively.

With this cost modeling, CAFO encodes input data to minimize the overall cost of performing a write request. It manipulates *N*-bit input data as an *n* by *m* matrix, where $N=n\times m$. Each row and column has an additional auxiliary bit that represents its corresponding row or column and is inverted. Also, CAFO labels the gain as *G*, which is calculated by *C*- $C_{inverted}$ where $C_{inverted}$ is the cost of writing an inverted form of the input data. Therefore, a positive gain means that inverted input data have lower overall costs than un-inverted data. Fig. 13 shows an example of a gain calculation when *a*, *b*, *c*, and *d* are 1, 2,

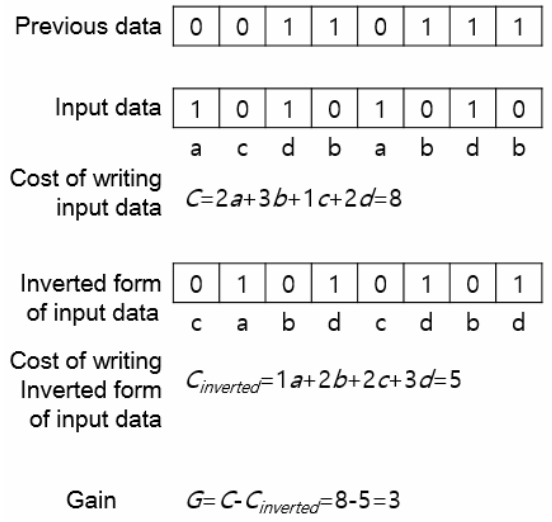


Fig. 13. Gain calculation when *a*, *b*, *c*, and *d* are 1, 2, 0, and 0, respectively.

```
1: Invert = True
2: ColumnCheck = False
3: while Invert do
4:   if PositiveGainRows() then
5:     Flip Rows with Positive Gain
6:   else
7:     if ColumnCheck then
8:       Invert = False
9:       break
10:    end if
11:  end if
12:  if PositiveGainColumns() then
13:    Flip Columns with Positive Gain
14:    ColumnCheck = True
15:  else
16:    Invert = False
17:  end if
18: end while
```

Fig. 14. Encoding algorithm of CAFO [17].

0, and 0, respectively. In the figure, gain G is 3, because C is 8, and $C_{inverted}$ is 5. In this case, the inverted form of the input data, i.e., 01010101, has lower cost than the input data, i.e., 10101010.

Fig. 14 presents the CAFO encoding algorithm. When PCM performs a write request, the cost, C, of each row is calculated by comparing input data and previous data in the row. After that, cost $C_{inverted}$ of each row is calculated by comparing the inverted form of input data and the previous data in the row. Finally, the gain of each row, called G, is calculated, and CAFO inverts the row if G is greater than 0. CAFO repeatedly searches and inverts rows and columns that have positive gains until there is no row and column to be inverted.

Fig. 15 presents the CAFO decoding algorithm. Each element of a data matrix has two auxiliary bits, i.e., one bit for the corresponding row and another bit for the corresponding column. If XORing these two bits results in 1, this means only one of the corresponding row or column is inverted. Therefore, the elements of a data matrix where auxiliary bits have different values are inverted when PCM reads data from memory cells.

```
// VR : Auxiliary bit for a row
// VC : Auxiliary bit for a column
// b[n][m] : Data block
1: for i=0, i++, i<n do
2:   for j=0, j++, j<m do
3:     if (VRi ++ VCj) then
4:       Read ~b[i][j]
5:     else
6:       Read b[i][j]
7:     end if
8:   end for
9: end for
```

Fig. 15. Decoding algorithm of CAFO [17].

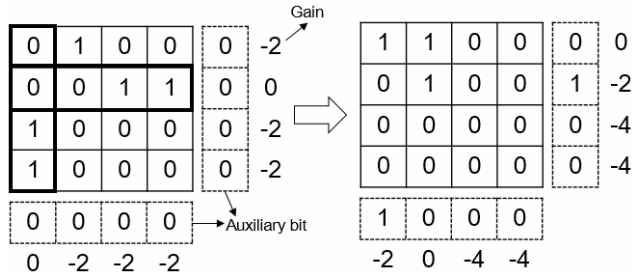


Fig. 16. Example of encoding optimization when *a*, *b*, *c*, and *d* are 1, 1, 0, and 0, respectively. A cell with 1 represents a bit value that is different from the previous value.

This paper also proposes optimization of the CAFO encoding algorithm. The encoding algorithm shown in Fig. 14 assumes the total write cost cannot be further reduced if every row and column has a zero or negative gain. However, it is possible to further reduce the cost of a write operation, even if no row or column has a positive gain. Fig. 16 shows an example of encoding optimization where a cell with 1 represents a bit that will be flipped because its value is different from the previous value. Although row 2 and column 1 do not have positive gains, inverting row 2 and column 1 further reduces the cost of the write operation from 5 to 3. Therefore, the condition that has to be met to further reduce the cost can be formulated as follows:

$$\sum_{r \in S} G_r + G_c - 2\sum_{r \in S} g_{r,c} > 0 \quad (4)$$

where S is the subset of rows, and $g_{r,c}$ is the gain from inverting the intersection cell at row r and column c. This formula is for inverting multiple rows and one column simultaneously. Inverting one row and multiple columns can be done by a formula similar to Eq. (4). In addition, since auxiliary bits are also stored in PCM cells, the cost of writing the auxiliary bits should be considered when inverting rows and columns.

Maddah, et al. [17] compared bit flip reduction in Flip-N-Write, FlipMin, and CAFO where they assumed them to have the same storage overhead, and also assumed *a*, *b*, *c*, and *d* to be 1, 1, 0, and 0, respectively. As a result, CAFO showed more bit flip reduction than Flip-N-Write and FlipMin when input data were 64B, 128B, and 512B. Also,

Table 3. Compared schemes and their storage overhead.

Scheme	Storage overhead
Conventional	0% (0 bits/64 bits)
DCW	0% (0 bits/64 bits)
Flip-N-Write	1.56% (1 bit/64 bits)
CAFO	25% (16 bits/64 bits)

Table 4. System configuration used for tracing a stress benchmark.

Structure	Configuration
CPU	x86-64 out-of-order core at 3.6 GHz
L1 D-Cache	128 KB, 8-way associative, 64 B line size
L1 I-Cache	128 KB, 8-way associative, 64 B line size
L2 Cache	2 MB, 8-way associative, 64 B line size
PCM	16 GB

CAFO showed a greater cost reduction than Flip-N-Write and FlipMin with various values of a, b, c, and d. In addition, even if the CAFO cost model was adopted in the compared schemes when encoding input data, CAFO achieved greater cost reduction than the other schemes.4. Experimental Setup

We evaluated lifetime extension from DCW, Flip-N-Write, and CAFO compared to conventional PCM. Table 3 shows the storage overhead of each scheme when the data bit width is 64 bits. DCW does not need any additional storage, because it just skips unnecessary bit updates without encoding data. Flip-N-Write uses a single flip bit to mark whether stored data are inverted or not. CAFO uses 16 auxiliary bits to handle 64-bit data as an eight-by-eight matrix.

Table 4 shows the system configuration for tracing a stress benchmark [7] that is a mixture of eight write-intensive benchmarks of SPEC CPU2006 [21] , i.e., milc, GemsFDTD, leslie3d, astar, soplex, zeusmp, omnetpp, and bwaves. The stress benchmarks were traced with MARSSx86 [22] on the system configurations described in Table 4.

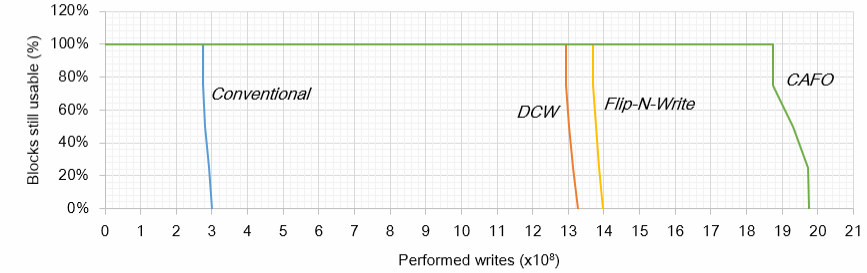
Table 5. The number of bit flips in each scheme after performing 10 million write requests from a stress benchmark.

Scheme	Bit flip ($x10^9$)	Reduction
Conventional	51.2	-
DCW	7.7	85.0%
Flip-N-Write	7.3	85.7%
CAFO	5.8	88.7%

In the experiment, we assumed memory access granularity of 64 bytes. A block was replaced by a spare block when it failed to write data [12, 16]. The write endurance of each memory cell followed a Gaussian distribution where the mean is 10^8 and the standard deviation is 10^7. Also, we used perfect wear-leveling that equally wears out PCM blocks so as to not consider the effect of the wear leveling algorithm assumed by Seong et al. [11]. Moreover, we modified DRAMSim2 [23] to examine the write endurance of each memory cell. Furthermore, since the lifetime estimation was significantly time-consuming, we applied the projection method described by Jacobvitz et al. [16] to shorten the simulation time.

5. Performance evaluation

Table 5 shows the number of bit flips in each scheme after 10 million write requests and their reduction rates, compared to conventional PCM. In conventional PCM, the number of bit flips reached 51.2 billion, since it flips all bits regardless of the previous data. DCW showed a significantly reduced number of bit flips because the input data of a stress benchmark frequently has all 0's or all 1's due to data initialization. In our experiment, these cases were more than 50% of the first 10 million write requests



Scheme	Performed writes before errors occur ($x10^8$)	Lifetime extension (Times)
Conventional	2.74	-
DCW	12.93	4.72
Flip-N-Write	13.69	5.00
CAFO	18.74	6.84

Fig. 17. Percentage of blocks still usable from the total blocks as write requests of a stress benchmark are performed. In addition, PCM lifetime extension in bit flip reduction schemes compared to the conventional PCM.

from the stress benchmark. Flip-N-Write further reduced bit flips by 85.7% by just adding one additional flip bit. CAFO showed an even more improved bit flip reduction rate, i.e., 88.7%.

Fig. 17 shows the number of blocks still usable out of the total number of memory blocks as write requests were performed. In the figure, each scheme for lifetime extension was calculated by the number of performed write requests until an error occurs, compared to conventional PCM. DCW improved PCM lifetime by 4.72 times, compared to conventional PCM. Flip-N-Write and CAFO further enhanced lifetime by 5.00 times and 6.84 times, respectively. Although CAFO showed the best performance in terms of bit flip reduction rate and lifetime extension, its storage overhead, i.e., 25%, was larger than the other schemes.

6. Summary

As more computing systems become multi-core processor systems, larger memory capacity is needed, and thus, PCM has gained attention as the promising alternative for main memory due to its high scalability. However, because of its write endurance problem, PCM cannot be directly adopted as main memory. Therefore, in this paper, we explored existing schemes to resolve the endurance problem by reducing the number of bit flips per cell.

First, we described the PCM operating principles in detail. PCM employs a phase change material called a chalcogenide alloy. The alloy has two different physical states: the crystal state and the amorphous state. These two states have different resistivity. The crystal state shows low resistivity (1kΩ), whereas the amorphous state shows high resistivity (1MΩ). With this resistivity difference, data can be stored in the cell. To change the cell to the crystal state, the SET current pulse, which has moderate power but long duration, heats the cell. Conversely, the RESET current pulse, which has high power but a short duration, changes the cell to the amorphous state. However, as the cell is repeatedly programmed, it becomes more unreliable, and eventually, cannot change states any more.

Then, we explored existing bit flip–reduction schemes. These schemes include DCW, Flip-N-Write, FlipMin, and CAFO. The DCW scheme skips bit programming of 0→0 and 1→1 because these situations do not need it (the previous bit value is the same as the bit value of the data to be stored). Flip-N-Write adds one additional bit, called a flip bit, to further reduce bit flips. When writing data, Flip-N-Write inverts input data and sets a flip bit when the Hamming distance between the input data and the previous data is over N/2 to limit the maximum number of bit flips under N/2. FlipMin uses coset coding to provide many vectors that encode candidates for the same input data. To generate vectors, FlipMin uses dual Hamming (72,64) code. When performing a write request, FlipMin maps 64-bit input data to parity with the Hamming code, and finds a preferred element from a set that has the same parity code. The CAFO scheme considers asymmetric PCM cells when encoding data to reduce the overall cost of performing a write request. It manipulates input data as a matrix, and continuously inverts a row or a column if inverting incurs less cost than before, until there is no row or column to be inverted.

In addition, these schemes were examined to compare performance in terms of bit flip–reduction rate and lifetime improvement. For bit flip–reduction rate, the number of bit flips in these schemes was compared after executing 10 million write requests of a stress benchmark. DCW, Flip-N-Write, and CAFO showed a significantly reduced number of bit flips, compared to conventional PCM, i.e., 85.0%, 85.7%, and 88.7%, respectively. Also, lifetime extension of each scheme was calculated by the number of write requests until an error occurs, compared to conventional PCM. As a result, DCW, Flip-N-Write, and CAFO improved the lifetime of PCM by 4.72, 5.00, and 6.84 times, respectively, compared to conventional PCM. Although CAFO showed the best performance in terms of bit flip reduction rate and lifetime extension, its storage overhead, i.e., 25%, was larger than other schemes.

References

[1] N. Aggarwal, et al., "Power-Efficient DRAM Speculation," *in Proc. of HPCA 2008*, pp. 317-328, Feb. 2008. Article (CrossRef Link)

[2] H. David, et al., "Memory power management via dynamic voltage/frequency scaling," *in Proc. of ICAC 2011*, pp. 31-40, June. 2011. Article (CrossRef Link)

[3] S. Chen, et al., "Rethinking Database Algorithms for Phase Change Memory," *in Proc. of CIDR 2011*, pp. 21-31, Jan. 2011. Article (CrossRef Link)

[4] B. Lee, et al., "Architecting Phase Change Memory As a Scalable Dram Alternative," *in Proc. of ISCA 2009*, pp. 2-13, June. 2009. Article (CrossRef Link)

[5] M. Prasanth, et al., "A Low-power Phase Change Memory Based Hybrid Cache Architecture," *in Proc. of GLSVLSI 2008*, pp. 395-398, May. 2008. Article (CrossRef Link)

[6] Y. Joo, et al., "Energy- and endurance-aware design of phase change memory caches," *in Proc. of DATE 2010*, pp. 136-141, Mar. 2010. Article (CrossRef Link)

[7] M. K. Qureshi, et al., "Enhancing lifetime and security of PCM-based Main Memory with Start-Gap Wear Leveling," *in Proc. of MICRO 2009*, pp. 14-23, Dec. 2009. Article (CrossRef Link)

[8] N. Seong, et al., "Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-change Memory with Dynamically Randomized Address Mapping," *in Proc. of ISCA 2010*, pp. 383-394, June. 2010. Article (CrossRef Link)

[9] H. Chang, et al., "Marching-Based Wear-Leveling for PCM-Based Storage Systems," *ACM TODAES*, Vol. 20, pp. 25–46, Mar. 2015. Article (CrossRef Link)

[10] S. Schechter, et al., "Use ECP, Not ECC, for Hard Failures in Resistive Memories," *in Proc. of ISCA 2010*, pp. 141-152, June. 2010. Article (CrossRef Link)

[11] N. H. Seong, et al., "SAFER: Stuck-At-Fault Error

Recovery for Memories," *in Proc. of MICRO 2010*, pp. 115-124, Dec. 2010. Article (CrossRef Link)
[12] D. H. Yoon, et al., "FREE-p: Protecting non-vvolatile memory against both hard and soft errors," *in Proc. of HPCA 2011*, pp. 466-477, Feb. 2011. Article (CrossRef Link)
[13] J. Fan, et al., "Aegis: Partitioning Data Block for Efficient Recovery of Stuck-at-faults in Phase Change Memory," *in Proc. of MICRO 2013*, pp. 433-444, Dec. 2013. Article (CrossRef Link)
[14] B. Yang, et al., "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme," *in Proc. of ISCAS 2007*, pp. 3014-3017, May. 2007. Article (CrossRef Link)
[15] S. Cho, et al., "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," *in Proc. of MICRO 2009*, pp. 347-357, Dec. 2009. Article (CrossRef Link)
[16] A. N. Jacobvitz, et al., "Coset coding to extend the lifetime of memory," *in Proc. of HPCA 2013*, pp. 222-233, Feb. 2013. Article (CrossRef Link)
[17] R. Maddah, et al., "CAFO: Cost aware flip optimization for asymmetric memories," *in Proc. of HPCA 2015*, pp. 320-330, Feb. 2015. Article (CrossRef Link)
[18] J. Chen, et al., "Exploring Dynamic Redundancy to Resuscitate Faulty PCM Blocks," *ACM JETC*, Vol. 10, pp. 31–53, June. 2014. Article (CrossRef Link)
[19] Z. Miao, et al., "Writeback-aware Partitioning and Replacement for Last-level Caches in Phase Change Main Memory Systems," *ACM TACO*, Vol. 8, pp. 53–73, Jan. 2012. Article (CrossRef Link)
[20] I. Engin, et al., "Dynamically Replicated Memory: Building Reliable Systems from Nanoscale Resistive Memories," *in Proc. of ASPLOS 2010*, pp. 3-14, Mar. 2010. Article (CrossRef Link)
[21] P. Aashish, et al., "Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite," *in Proc. of ISCA 2007*, pp. 412-423, June. 2007. Article (CrossRef Link)
[22] A. Patel, et al., "MARSS: A full system simulator for multicore x86 CPUs," *in Proc. of DAC 2011*, pp. 1050-1055, June. 2011. Article (CrossRef Link)
[23] P. Rosenfeld, et al., "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, Vol. 10, pp. 16–19, Jan. 2011. Article (CrossRef Link)
[24] W. Zhang, et al., "Characterizing and mitigating the impact of process variations on phase change based memory systems," *in Proc. of MICRO 2008*, pp. 2-13, Dec. 2009. Article (CrossRef Link)
[25] K. Kim, et al., "Reliability investigations for manufacturable high density PRAM," *in Proc. of IRPS 2005*, pp. 157-162, Apr. 2005. Article (CrossRef Link)
[26] S. Raoux, et al., "Phase change materials and phase change memory," *Materials Research Society*, Vol. 39, pp. 703–710, Aug. 2014. Article (CrossRef Link)
[27] I. Heng, et al., "Error correcting codes associated with complex Hadamard matrices," *Applied Mathematics Letters*, Vol. 11, pp. 77–80, July. 1998. Article (CrossRef Link)
[28] G.D. Forney, "Coset codes. I. Introduction and geometrical classification," *IEEE Transactions on Information Theory*, Vol. 34, pp. 1123–1151, Sep. 1988. Article (CrossRef Link)
[29] G.D. Forney, "Coset codes. II. Binary lattices and related codes," *IEEE Transactions on Information Theory*, Vol. 34, pp. 1152–1187, Sep. 1988. Article (CrossRef Link)

Miseon Han received her BEng in Electrical Engineering from Korea University, Seoul, Rep. of Korea, in February 2012. Currently, she is a PhD candidate in the same department. Her research interests include compiler support, microarchitecture, and memory designs.

Youngsun Han received his BEng and PhD from the School of Engineering at Korea University, Seoul, Rep. of Korea, in 2003 and 2009, respectively. He was a senior engineer at Samsung LSI from 2009 to 2011. Currently, he is an assistant professor with the Department of Electronic Engineering, Kyungil University. His research interests include system software optimization, microarchitectures, high-performance computing, and SoC design.