Extending the Lifetime of NVMs with Compression

Jie Xu, Dan Feng, Yu Hua, Wei Tong*, Jingning Liu and Chunyan Li
Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System
(School of Computer Science and Technology, Huazhong University of Science and Technology)
Ministry of Education of China
*Corresponding author: Wei Tong (Tongwei@hust.edu.cn)
Email:{xujie_dsal, dfeng, csyhua, Tongwei, jnliu, lichunyan}@hust.edu.cn

***Abstract*—Emerging Non-Volatile Memories (NVMs) such as Phase Change Memory (PCM) and Resistive RAM (RRAM) are promising to replace traditional DRAM technology. However, they suffer from limited write endurance and high write energy consumption. Encoding methods such as Flip-N-Write, FlipMin and CAFO can reduce the bit flips of NVMs by exploiting additional capacity to store the tag bits of encoding methods. The effects of encoding methods are limited by the capacity overhead of the tag bits. In this paper, we propose COE to COmpress cacheline for Extending the lifetime of NVMs. COE exploits the space saved by compression to store the tag bits of data encoding methods. Through combining data compression techniques with data encoding methods, COE can reduce the bit flips with negligible capacity overhead. We further observe that the saved space size of each compressed cacheline varies, and different encoding methods have different tradeoffs between capacity overhead and effects. To fully exploit the space saved by compression for improving lifetime, we select the proper encoding methods according to the saved space size. Experimental results show that our scheme can reduce the bit flips by 14.2%, decrease the energy consumption by 11.8% and improve the lifetime by 27.5% with only 0.2% capacity overhead.**

I. Introduction

Non-Volatile Memories (NVMs) such as Phase Change Memory (PCM) and Resistive RAM (RRAM) have emerged as potential replacement candidates of DRAM technology due to their non-volatility, high density and low read latency. However, they suffer from high write energy and limited write endurance. The write energy of PCM and RRAM are several times more than that of DRAM. Besides, the endurances of PCM and RRAM are 10^8 and 10^{10} respectively [1][2][3], which are several orders of magnitude fewer than DRAM (10^{16}).

Some existing works [4][5][6][7][8] proposed to reduce the bit flips (i.e., $1 \rightarrow 0$ and $0 \rightarrow 1$) through data encoding. Flip-N-Write [6] encoded the new data bits by giving every N data bits one tag bit. If the bit flips of writing the new data and its tag bit exceed $(N+1)/2$, the new data bits will be flipped and the tag bit is set to 1. The reduction of bit flips in Flip-N-Write is limited by the capacity overhead of tag bits. If we give fewer data bits one tag bit, Flip-N-Write can reduce more bit flips. Unfortunately, the tag bits will incur significant capacity overhead if we give fewer data bits one tag bit. For example, Flip-N-Write can reduce the bit flips by 25% with 50% capacity overhead (1 tag bit for every 2 data bits), while the reduction is decreased to 14.6% with 6.25% capacity overhead (1 tag bit for every 16 data bits). Other encoding methods have the same property. FlipMin [4] can reduce the bit flips by 31.2% with 100% capacity overhead. The decrease of bit flips falls to 24.5% if the capacity overhead is 12.5%. Although these methods can reduce the bit flips, the capacity overhead cannot be ignored.

This work aims to reduce the bit flips with negligible capacity overhead. Data compression techniques such as Frequent Pattern Compression (FPC) [9] and Base Delta Immediate (BDI) [10] can reduce the size of the data to store. On the other side, data encoding methods require additional capacity to store the tag bits. We propose to exploit the space saved by data compression techniques to store the tag bits of data encoding methods. Moreover, we observe that the size of the space saved by each compressed cacheline is different. Some highly compressed cacheline can offer more capacity, while slightly compressed cacheline can offer less capacity. Besides, different encoding methods have different tradeoffs between capacity and effects. To fully exploit the space saved by compression, we use the encoding methods with high capacity overhead for those highly compressed cachelines. We have the following contributions in this paper:

- We propose to exploit the space saved by compression to store the tag bits of data encoding methods.
- To fully exploit the saved space for bit flips reduction, we select the best-performing encoding method according to the size of the space saved by compression.
- Experimental results show that our scheme can reduce the bit flips by 14.2%, decrease the energy consumption by 11.8% and improve the lifetime by 27.5% with only 0.2% capacity overhead.

The rest of this paper is structured as follows. Section II introduces the background and related work. Section III describes the motivation. Section IV presents the design and implementation. Sections V and VI describe the experiment setup and conclusion.

II. Background and related work

A. Background

NVMs such as PCM and RRAM suffer from high write energy and limited write endurance.

1) PCM: PCM exploits phase change material such as $Ge_2Sb_2Te_2$ (GST) to store digital bits. When the GST is

978-3-9819263-0-9/DATE18/©2018 EDAA

in crystalline state, the resistance of PCM cell is low, and this low resistance state represents logical value ‘1’. When the GST is in high resistance state (amorphous state), PCM stores ‘0’. The PCM cell will be reset if it is heated above $600°C$ for a short duration. In contrast, a long duration but small amplitude current is applied to set a PCM cell. The repeated heat stress will damage the phase change material, and a PCM cell may get stuck at the amorphous or the crystalline state after $10^6 \sim 10^9$ writes [11]. The endurance of PCM is several order of magnitudes less than DRAM (10^{16}). Besides, PCM suffers from high write energy. The write energy of PCM is $20pJ/bit$, which is twice more than DRAM.

2) RRAM: A RRAM cell consists of a top electrode and a bottom electrode, and a metal-oxide layer (HfO_2, Ta_2O_5, etc.) [12] between them. The logical values are stored in RRAM by changing the resistance of the RRAM cells. The high resistance state (HRS) is used to represent logical value ‘0’, and low resistance state (LRS) represents ‘1’. In order to change the resistance of a RRAM cell, an external voltage (V_{set} or V_{reset}) is applied across the cell. A RRAM cell can endure 10^{10} writes [3][13] in the best existing architectures. The write energy of RRAM is also several times more than DRAM.

B. Related work

Many works have been proposed to reduce the bit flips. Existing works can be divided into the following two categories.

1) Reducing bit flips with data encoding: Encoding methods encoded the new data bits into the vector with the minimum bit flips and used additional capacity to store the tag bits. Flip-N-Write [6] flipped the new data bits if more than half of the data bits needs to be written. CAFO [5] modeled the memory block as $N * M$ array and used Flip-N-Write in both the rows and columns to minimize the bit flips. Captopril [8] extended Flip-N-Write to reduce bit flips in hot locations. Different from Flip-N-Write, FlipMin [4] used coset code to generate vectors and selected the vector that had the minimum bit flips. Pseudo-Random [7] was designed based on coset coding, and it mapped the data bits into highly random data vectors. ES [14] extended the methods for Single-Level Cell (SLC) to Multi-Level Cell (MLC) magnetic memories through encoding the hard bits and soft bits separately.

2) Data compression with data encoding: Some other works proposed to combine the data encoding with data compression. David B. Dgien [15] proposed to compress the data bits before write operations. AFNW [16] extended Flip-N-Write by adapting the tag bits to the compressed data bits. The saved space is wasted in Refs. [15][16]. DIN [17] compressed the cacheline and used the saved space to mitigate write disturbance. Amin Jadidi [18] exploited the space saved by compression for hard-error tolerance and wear leveling. For Triple-Level Cell (TLC) NVMs, CRADE [19] and CompEx [20] integrated data compression with expansion coding to reduce write energy and latency. Some other works [21][22][23][24] proposed to dynamically configure the MLC as SLC, Tri-state Cell or MLC according to the size of the space saved by compression. None of the prior works attempts to exploit the saved space for bit flips reduction.

III. Motivation

Data compression techniques can reduce the size of the data to store, and data encoding methods can expand the size for reducing bit flips. Compression techniques can work in collaboration with encoding method to reduce bit flips.

A. The size of the space saved by compression

Various compression techniques have been proposed. We evaluate FPC [9] in this work because of its low implementation overhead. Each compressed word requires a 3-bit prefix to indicate the data pattern and the size of the compressed word. To reduce the overhead of prefixes, FPC is extended for 64-bit words in this work. Table I lists the data patterns that 64-bit FPC can compress [9][20]. A 64-bit word can be compressed to 0, 8, 16 or 32 bits. For a cacheline which consists of eight words, the compressed size of each word is different, and the size of the compressed cacheline is also different. The total number of bits in the compressed cacheline may range from 0 to 512. Fig. 1 shows the distribution of compressed cacheline sizes for thirteen benchmarks selected from SPEC CPU 2006 [25]. The size of the space saved by compression is different due to the variation of the compressed cacheline size. Some compressed cachelines are smaller than a word, and the space of seven words is saved. Some other compressed cachelines have more than seven words, and the saved space is less than one word.

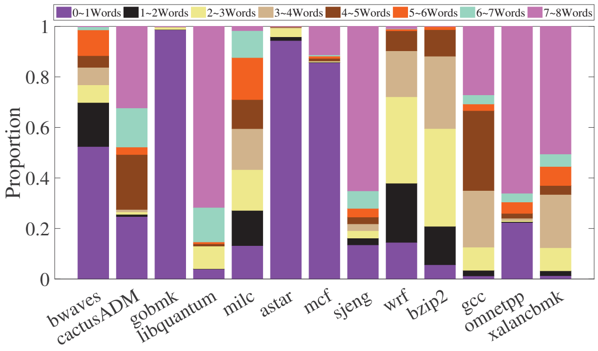


Fig. 1. The distribution of the compressed cacheline size.

B. Tradeoffs between effects and capacity overhead in encoding methods

Various encoding methods have been proposed. We discuss the tradeoffs in Flip-N-Write [6] and FlipMin [4] due to their simplicity and high performance. Other encoding methods such as CAFO [5] mainly addressed the asymmetry in programming memory cells. CAFO is not implemented in this work because we assume the cost of writing 0 and 1 is the same.

1) Flip-N-Write: In Flip-N-Write [6], a cacheline is divided into M words, and each word has N bits. For each N-bit word, a tag bit is given to indicate whether the word is flipped or not. The tag bit is initialized to 0 before encoding. If the bit flips of writing the new data and its tag bit exceed $(N+1)/2$, the new

TABLE I
THE DATA PATTERNS OF 64-BIT FPC [9][20].

Prefix	Pattern encoded	Example	Compressed	Encoded size
000	Zero run	0x0000000000000000	0x0	0 bits
001	8-bits sign-extended	0x000000000000007F	0x7F	8 bits
010	16-bits sign-extended	0xFFFFFFFFFFFFB6B6	0xB6B6	16 bits
011	Half-word sign-extended	0x0000000076543210	0x76543210	32 bits
100	Half-word, padded with a zero half-word	0x7654321000000000	0x76543210	32 bits
101	Two half-words, each two bytes sign-extended	0xFFFFBEEF00003CAB	0xBEEF3CAB	32 bits
110	Consisting of four repeated double bytes	0xCAFECAFECAFECAFE	0xCAFE	16 bits
111	Uncompressible	0x0123456789ABCDEF	0x0123456789ABCDEF	64 bits

data bits are flipped, and the tag bit is set to 1. The reduction of bit flips will decrease if N increases. If N equals 2, the effect of Flip-N-Write is the best, and Flip-N-Write can reduce 25% more bit flips than DCW [26]. The reduction drops to 14.6% if N equals 16. However, the capacity overhead of Flip-N-Write increases significantly if N decreases. The capacity overhead is 50% if N equals 2, and the overhead falls to 6.25% if N equals 16. The encoding and decoding processes of Flip-N-Write are very simple, and the latency overhead of Flip-N-Write is negligible.

2) FlipMin: FlipMin [4] used coset code to minimize the bit flips. Each dataword is mapped to a coset of vectors. FlipMin chooses the vector that has the minimum bit flips. The bit flips reduction of FlipMin increases with more capacity overhead. FlipMin can reduce the bit flips by 31.2% with 100% capacity overhead, and the reduction is decreased to 24.5% with 12.5% capacity overhead. The latency overhead of encoding and decoding in FlipMin varies significantly. The coset code with 100% capacity overhead (RM(1, 3)) incurs $4.09ns$ encoding latency and $0.38ns$ decoding latency, while the coset code with 12.5% capacity overhead (RM(1, 7)) incurs $12.86ns$ encoding latency and $0.59ns$ decoding latency.

Besides, the tradeoffs exist between different encoding methods, e.g., Flip-N-Wite and FlipMin. The comparison of Flip-N-Write and FlipMin is shown in Table II. Flip-N-Write can use at most 50% additional capacity as tag bits, while FlipMin can use more than 100% additional capacity. In the aspects of latency and effect, the compromises also exist. FlipMin can reduce 6.5% more bit flips than Flip-N-Write with the same capacity overhead (12.5%). However, the encoding latency of FlipMin is several times more than Flip-N-Write.

TABLE II
COMPARISON OF FLIP-N-WRITE AND FLIPMIN [6] [4]

		Encoding Latency	Decoding Latency	Bit flips Reduction	Capacity Overhead
Flip-N-Write	N=2	<1ns	<0.1ns	25.0%	50%
	N=4	<1ns	<0.1ns	21.9%	25%
	N=8	<1ns	<0.1ns	18.3%	12.50%
	N=16	<1ns	<0.1ns	14.6%	6.25%
FlipMin	RM(1,3)	4.09ns	0.38ns	31.25%	100%
	RM(1,7)	12.86ns	0.59ns	24.50%	12.50%

IV. DESIGN AND IMPLEMENTATION

A. Design

1) Overview: The main idea of our scheme is to exploit the space saved by compression to store the tag bits of data encoding methods. For each cacheline, we first use compression technique to compress the cacheline. If the cacheline is uncompressible, we will not encode it because no extra space is offered. If the cacheline is compressible, we first compress it. Then we use the saved space to store tag bits. A suitable encoding method is chosen according to the saved space size. Our scheme works as Fig. 2 describes.

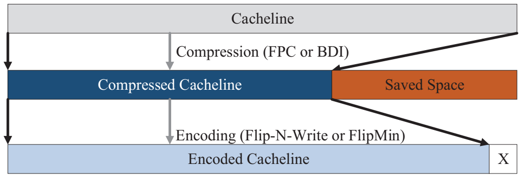


Fig. 2. Overview of our scheme.

2) The organization of prefixes and data: We use a tag bit (compression tag) to indicate whether the cacheline is compressible or not. If there is at least one compressible word, the cacheline is considered as compressible and the tag bit is set. Otherwise the cacheline is uncompressible and the tag bit is reset. If the cacheline is compressible, the saved space size is 32 bits at least. For each word of the compressible cacheline, we use a 3-bit prefix to indicate the data pattern. The total number of bits of prefixes in compressible cacheline is 24, which is smaller than 32. Therefore, the size of the compressible cacheline with 24-bit prefix will not exceed 512. If the cacheline is uncompressible, we do not use prefixes and each word is stored without any modification. The additional capacity overhead is only one bit. Prior works [20][16] proposed to store the prefix with each compressed word. The prefix of a word is stored after the prior word. We have to decompress the word one by one in this way, and the sequential decompression of each word will cause accumulated decompression latency. We propose to place the prefixes together, as shown in Fig. 3. The location of each compressed word can be located by the first 24-bit prefixes with parallel decompression.

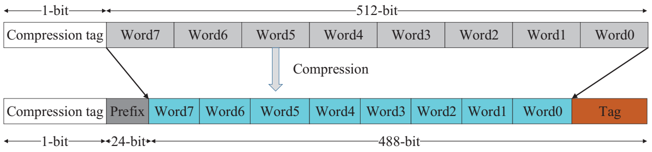


Fig. 3. The organization of prefixes and eight compressed words for the compressible cacheline.

3) The encoding methods: We use Flip-N-Write or FlipMin to encode the compressed cacheline. When selecting from

Flip-N-Write and FlipMin, we preferentially choose Flip-N-Write because of its low encoding/decoding latency. However, the tag bits of Flip-N-Write is at most 50% of the data bits. Flip-N-Write cannot fully exploit the saved space if the saved space is more than half of the data bits. Therefore, we use FlipMin if the saved space is adequate. FlipMin has some variations, and we use the FlipMin with 100% capacity due to its high performance. We use S to represent the size of the space saved by compression and use D to represent the compressed data size. For a 64-bit word compressed to 8 bits, S equals 53 (the prefix occupies additional 3 bits) and D equals 8. For the cacheline, S stands for the saved space size of the eight words, and D is the total number of bits in the eight compressed words. If S/D is smaller than 50%, we use Flip-N-Write and select a best-performing Flip-N-Write according to the saved space size. If S/D is between 50% and 100%, we give every 2 data bits 1 tag bit in Flip-N-Write. If S/D is greater than 100%, we apply FlipMin to the data bits. The relationship between the effects of the encoding methods and the saved space size is illustrated in Fig. 4.

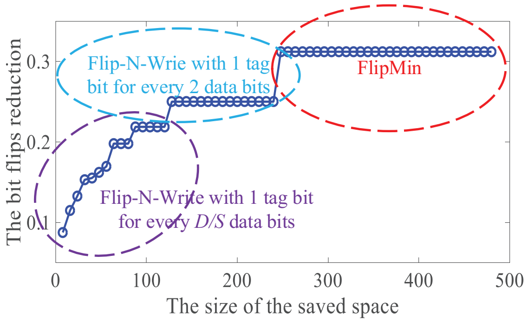
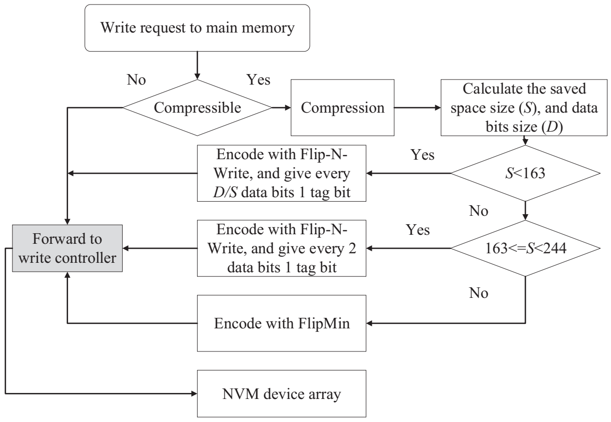


Fig. 4. The relationship between the saved space size and the bit flips reduction.

B. Implementation

The implementation of our design includes Encoder and Decoder. The Encoder and Decoder are on the write path and read path respectively.

1) Encoder: The Encoder consists of two parts, i.e., compression and encoding. When the NVM controller receives a write request, each of the eight words are compared with the data patterns illustrated in Table I to attempt compression. If none of the eight words is compressible, this cacheline is uncompressible, and the compression tag is reset. The uncompressible cacheline is sent to the write controller without compression or encoding. If at least one of the eight word is compressible, this cacheline is compressible, and the compression tag is set. For the compressible cacheline, the cacheline is compressed first, and then the saved space size (S) and compressed data bits size (D) are calculated. Different encoding methods are applied to the compressed cacheline based on the value of S. If S is smaller than 163, we choose Flip-N-Write and give every D/S data bits 1 tag bit. If S is between 163 and 244, we choose Flip-N-Write with 1 tag bit for every 2 data bits. If S is greater than 244, we select the FlipMin with 100% capacity overhead. The encoding process of Encoder is shown in Fig. 5.



Write request to main memory
No
Yes
Compressible
Compression
Calculate the saved space size (S), and data bits size (D)
Encode with Flip-N-Write, and give every D/S data bits 1 tag bit
Yes
S<163
No
Forward to write controller
Encode with Flip-N-Write, and give every 2 data bits 1 tag bit
Yes
163<=S<244
No
Encode with FlipMin
NVM device array

Fig. 5. The process of the Encoder.

2) Decoder: Since the cacheline has been compressed and encoded by the Encoder, the Decoder consists of decoding and decompression. When the NVM module receives a read request from the memory controller, the Decoder works as follows. First, use the compression tag to determine that whether the cacheline is compressed or not. If the cacheline is not compressed, it is sent to the read buffer directly. For the compressed cacheline, the 24-bit prefixes are exploited to calculate S and D. Then, the encoding method is determined according to S, and the corresponding decoding method is applied to the encoded data bits. After decoding, the data bits are decompressed by FPC word by word. Assuming that the encoding method is FlipMin, the Decoder will work as Fig. 6 illustrates.

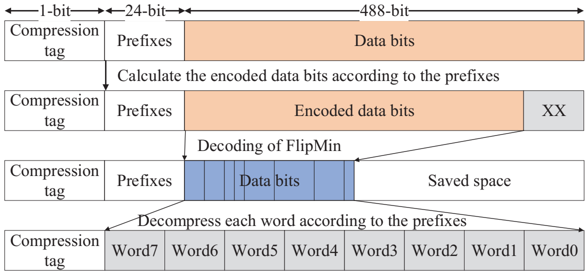


Fig. 6. The process of the Decoder.

C. Overhead

The Encoder and Decoder incur additional latency and energy overhead only if the cacheline is compressible. The implementation of the Decoder and Encoder consists of three parts, i.e., compression and decompression, encoding and decoding of Flip-N-Write or FlipMin, and Multiplexer which calculates the saved space size and selects an encoding method.

1) Compression: The hardware, energy and latency overhead of FPC has been discussed in Refs. [9][17][21]. The hardware overhead is similar to $16K$ PCM cells. The encoding and decoding energy is $2.1pJ$ and $1.2pJ$ [21]. The latencies of compression and decompression are estimated to be $2ns$ and $1ns$ [20] in this work.

2) Encoding: The encoding and decoding latencies of Flip-N-Write and FlipMin are shown in Table II. The energy overhead of Flip-N-Write is negligible [6], and the decoding/encoding energy of FlipMin is $0.3pJ/8.4pJ$ [4].

3) Multiplexer: The Multiplexer works in both Decoder and Encoder. To evaluate the overhead of the Multiplexer, we use Synopsys Design Compiler to synthesize the logic in $130nm$ technology, and we scale the results down to $22nm$ technology node. The latency of the Multiplexer is $1.52ns$, and energy overhead is $1.7pJ$. The hardware overhead of Multiplexer is 881 logic gates.

V. Experimental Setup

We use Gem5 [27] to evaluate our schemes, and the main memory model is based on NVMain [28]. NVMain is a cycle-level main memory simulator designed to simulate emerging NVMs at the architectural level. The configuration of the target system is given in Table III. The system is based on a four-core processor. Fourteen benchmarks are used in our experiment. All these benchmarks are selected from SPEC CPU 2006 [25].

TABLE III
System configurations

Cores	4-Core, 2.0GHz, out-of-order
L1 I/D cache	private, 32KB, 2-way, 2-cycle latency
L2 Cache	private, 1MB, 8-way, 20-cycle latency
L3 Cache	shared, 16MB, 16-way, 50-cycle latency
Memory Controller	FRFCFS
Memory Organization	4GB PCM, Read 50ns, Write 150ns

We evaluate the following seven different schemes. All the schemes use DCW to reduce the redundant bit flips.

- DCW [26]: The redundant bit flips are eliminated.
- FPC [9]: The cacheline is compressed by FPC at the granularity of 64-bit words.
- AFNW [16]: The tag bits are assigned to the compressed data bits. Each 64-bit word has 4 tag bits in AFNW.
- Flip-N-Write [6]: The data bits are flipped if flipping can reduce the bit flips. We give every 8 data bits 1 tag bit.
- FlipMin [4]: We use the RM(1,3) to generate the coset in this work.
- COE: The space saved by compression is exploited to store the tag bits of Flip-N-Write. Only Flip-N-Write is used to encode the compressed data.
- COEF: The space saved by compression is exploited to store tag bits of Flip-N-Write or FlipMin. FlipMin or Flip-N-Write is used to encode the compressed data according to the saved space size.

A. Experimental results

The proposed designs are evaluated in terms of bit flips, energy and lifetime. All the experimental results are normalized to DCW [26]. The overview of the comparison of capacity overhead, bit flips, energy and lifetime is shown in Table IV.

1) Bit flips: Fig. 7 illustrates the normalized bit flips for each benchmark. The average bit flips reduction of FPC, AFNW, Flip-N-Write, FlipMin, COE and COEF is -3.9%, 7.3%, 14.2%, 48.3%, 6.5% and 14.2%. The same as Refs. [29][18], FPC leads to more bit flips than DCW. The reason is that FPC will destroy the data similarity and redundancy. Dirty cacheline may have clean words, and DCW can eliminate the writes to those clean words. The clean words after compression will be different from the original words, and compression will lead to more bit flips in this case. Besides, FPC needs 24-bit prefixes per cacheline to indicate the data patterns, and the writes to prefixes will result in additional bit flips. FlipMin can reduce the most bit flips because it exploits coset code to generate a coset of vectors and costs 100% additional capacity. COE has the similar effect to AFNW. However, AFNW incurs 6.05% more capacity overhead than COE. Comparing COEF with COE, COEF can reduce 7.7% more bit flips than COE. COEF exploits both Flip-N-Write and FlipMin to encode and can fully exploit the space saved by compression. COEF has the same bit flips reduction as Flip-N-Write, and COEF can save 12.3% more capacity than Flip-N-Write.

TABLE IV
The comparison of capacity overhead, bit flips, energy and lifetime.

	Capacity overhead	Bit flips	Energy	Lifetime
DCW	0	1	1	1
FPC	1.56%	1.04	1.03	0.99
AFNW	6.25%	0.93	0.94	1.20
Flip-N-Write	12.50%	0.86	0.89	1.33
FlipMin	100%	0.52	0.58	3.94
COE	0.20%	0.93	0.95	1.12
COEF	0.20%	0.86	0.88	1.28

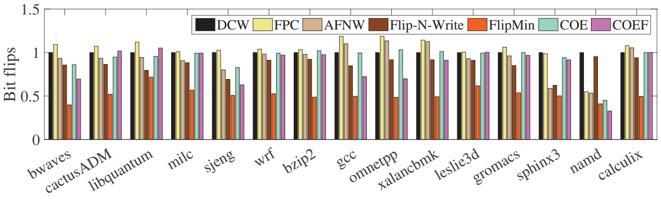


Fig. 7. The comparison of the normalized bit flips.

2) Energy: The total energy consumption of different schemes is shown in Fig. 8. For NVMs, write energy consumption dominates in the total energy consumption. Therefore, the energy reduction is the same as the bit flips reduction in most benchmarks. The total energy is reduced by -2.7%, 5.6%, 10.7%, 41.9%, 5.0% and 11.8% in FPC, AFNW, Flip-N-Write, FlipMin, COE and COEF. The total energy reduction is nearly the same as the bit flips reduction in all the benchmarks except libquantum and sjeng. For libquantum and sjeng, large number of words in dirty cachelines are clean, and therefore the write energy is less than other operations such as reads and activates. The total energy consumption is almost the same in different schemes for libquantum and sjeng.

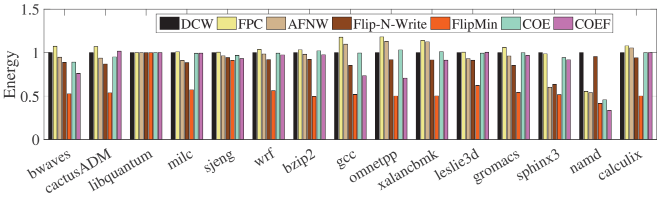


Fig. 8. The comparison of the normalized energy.

3) Lifetime: We use the total number of bit flips to estimate the lifetime. The lifetime is defined as $\frac{Capacity}{Bitflips}$, and lifetime is improved due to the decrease of bit flips and the use of additional capacity. Our schemes can reduce the bit flips, and therefore the lifetime is improved. The lifetime improvement is -0.7%, 19.8%, 32.8%, 294.4%, 11.7% and 27.5% in FPC, AFNW, Flip-N-Write, FlipMin, COE and COEF respectively. FlipMin can improve the lifetime by 294.4% because it can reduce 41.9% bit flips and has extra 100% capacity overhead. COEF can reduce the same bit flips as Flip-N-Write. However, the lifetime improvement is less than Flip-N-Write because Flip-N-Write has additional 12.5% capacity overhead.

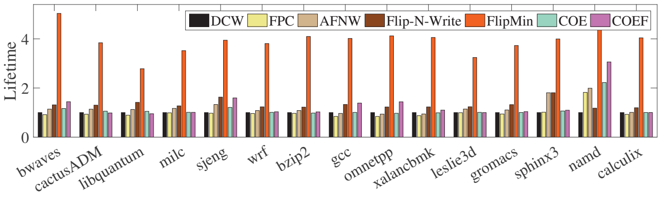


Fig. 9. The comparison of the normalized lifetime.

VI. Conclusion

NVMs suffer from limited write endurance and high write energy. This paper proposes to extend the lifetime of NVMs by combining the data compression techniques and data encoding methods. We exploit the space saved by compression to store the tag bits of data encoding methods. Based on the observations that the size of the space saved by each compressed cacheline varies and different encoding methods have different tradeoffs between capacity overhead and effects, we select the best-performing data encoding methods according to the size of the space saved by compression. Experimental results show that our schemes can reduce the bit flips by 14.2%, decrease the energy consumption by 11.8% and improve the lifetime by 27.5% with only 0.2% capacity overhead.

Acknowledgment

This work was supported by the National High Technology Research and Development Program (863 Program) No.2015AA015301, the National Natural Science Foundation of China (NSFC), No.61772222, No.61772212, No.61472153, No.61502191, State Key Laboratory of Computer Architecture, No.CARCH201505. This work was also supported by Engineering Research Center of Data Storage Systems and Technology, Ministry of Education, China.

References

[1] S. Mittal, J. S. Vetter, and D. Li, “A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1524–1537, 2015.

[2] H. A. Khouzani, F. S. Hosseini, and C. Yang, “Segment and conflict aware page allocation and migration in dram-pcm hybrid main memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1458–1470, 2017.

[3] S. Shirinzadeh, M. Soeken *et al.*, “Endurance management for resistive logic-in-memory computing architectures,” in *Proceedings of DATE*. IEEE, 2017, pp. 1092–1097.

[4] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, “Coset coding to extend the lifetime of memory,” in *Proceedings of HPCA*. IEEE, 2013, pp. 222–233.

[5] R. Maddah, S. M. Seyedzadeh, and R. Melhem, “Cafo: Cost aware flip optimization for asymmetric memories,” in *Proceedings of HPCA*. IEEE, 2015, pp. 320–330.

[6] S. Cho and H. Lee, “Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance,” in *Proceedings of MICRO*. IEEE, 2009, pp. 347–357.

[7] S. M. Seyedzadeh, R. Maddah *et al.*, “Pres: Pseudo-random encoding scheme to increase the bit flip reduction in the memory,” in *Proceedings of DAC*. IEEE, 2015, pp. 1–6.

[8] M. Jalili and H. Sarbazi-Azad, “Captopril: Reducing the pressure of bit flips on hot locations in non-volatile main memories,” in *Proceedings of DATE*. IEEE, 2016, pp. 1116–1119.

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

[10] G. Pekhimenko, V. Seshadri *et al.*, “Base-delta-immediate compression: Practical data compression for on-chip caches,” in *Proceedings of PACT*. ACM, 2012, pp. 377–388.

[11] H. A. Khouzani, Y. Xue, and C. Yang, “Fully exploiting pcm write capacity within near zero cost through segment-based page allocation,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 4, p. 31, 2016.

[12] W. Zhao, J. M. Portal *et al.*, “Design and analysis of crossbar architecture based on complementary resistive switching non-volatile memory cells,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 6, pp. 2484–2496, 2014.

[13] H. Lee, Y. Chen *et al.*, “Evidence and solution of over-reset problem for hfo x based resistive memory with sub-ns switching speed and high endurance,” in *Proceedings of IEDM*. IEEE, 2010, pp. 19–7.

[14] J. Xu, D. Feng *et al.*, “Encoding separately: An energy-efficient write scheme for mlc stt-ram,” in *Proceedings of ICCD*. IEEE, 2017, pp. 581–584.

[15] D. B. Dgien, P. M. Palangappa, N. A. Hunter, J. Li, and K. Mohanram, “Compression architecture for bit-write reduction in non-volatile memory technologies,” in *Proceedings of NANOARCH*. ACM, 2014, pp. 51–56.

[16] P. M. Palangappa and K. Mohanram, “Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories,” in *Proceedings of GLSVLSI*. ACM, 2015, pp. 221–224.

[17] L. Jiang, Y. Zhang, and J. Yang, “Mitigating write disturbance in super-dense phase change memories,” in *Proceedings of DSN*. IEEE, 2014, pp. 216–227.

[18] A. Jadidi, M. Arjomand *et al.*, “Exploring the potential for collaborative data compression and hard-error tolerance in pcm memories,” in *Proceedings of DSN*. IEEE, 2017, pp. 85–96.

[19] J. Xu, D. Feng *et al.*, “Improving performance of tlc rram with compression-ratio-aware data encoding,” in *Proceedings of ICCD*. IEEE, 2017, pp. 573–580.

[20] P. M. Palangappa and K. Mohanram, “Compex: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvm,” in *Proceedings of HPCA*. IEEE, 2016, pp. 90–101.

[21] L. Jiang, B. Zhao *et al.*, “Improving write operations in mlc phase change memory,” in *Proceedings of HPCA*. IEEE, 2012, pp. 1–10.

[22] M. Arjomand, A. Jadidi *et al.*, “A morphable phase change memory architecture considering frequent zero values,” in *Proceedings of ICCD*. IEEE, 2011, pp. 373–380.

[23] L. Jiang, Y. Zhang, and J. Yang, “Er: Elastic reset for low power and long endurance mlc based phase change memory,” in *Proceedings of ISLPED*. ACM, 2012, pp. 39–44.

[24] H. G. Lee, S. Baek *et al.*, “A compression-based hybrid mlc/slc management technique for phase-change memory systems,” in *Proceedings of ISVLSI*. IEEE, 2012, pp. 386–391.

[25] J. L. Henning, “Spec cpu2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[26] B.-D. Yang, J.-E. Lee *et al.*, “A low power phase-change random access memory using a data-comparison write scheme,” in *Proceedings of ISCAS*. IEEE, 2007, pp. 3014–3017.

[27] N. Binkert, B. Beckmann *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[28] M. Poremba, T. Zhang, and Y. Xie, “Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems,” *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[29] J. Kong and H. Zhou, “Improving privacy and lifetime of pcm-based main memory,” in *Proceedings of DSN*. IEEE, 2010, pp. 333–342.