A Low-Overhead Encoding Scheme to Extend the Lifetime of Nonvolatile Memories

Dan Feng, *Member, IEEE*, Jie Xu, Yu Hua, *Senior Member, IEEE*, Wei Tong, Jingning Liu, Chunyan Li, and Yiran Chen, *Fellow, IEEE*

***Abstract*—Emerging nonvolatile memories (NVMs) are promising to replace DRAM as main memory. However, NVMs suffer from limited write endurance and high write energy. Encoding method reduces the bit flips of NVMs by exploiting additional tag bits to encode the data. The effect of the encoding method is limited by the capacity overhead of the tag bits. In this article, we propose to exploit the space saved by compression to store the tag bits of the encoding method. We observe that the saved space size of each compressed cache line varies, and different encoding methods have different tradeoffs between capacity overhead and effect. To fully exploit the space saved by compression for improving lifetime, we select the proper encoding method according to the saved space size. To improve the compression coverage and compression ratio, we select an efficient compression scheme from two compression algorithms and provide more space for data encoding. Still, some data patterns cannot be compressed by any compression technique. We use the Flip-N-Write with 3.1% capacity overhead to encode uncompressible cache lines. The experimental results show that our scheme reduces the bit flips by 32.5%, decreases the energy consumption by 22.6% and improves the lifetime by 69.9% with 3.5% capacity overhead.**

***Index Terms*—Bit flips, compression, encoding, lifetime, nonvolatile memories (NVMs).**

I. Introduction

NONVOLATILE memories (NVMs), such as phase change memory (PCM) and resistive RAM (RRAM) have emerged as potential replacement candidates of the DRAM technology due to their nonvolatility, high density, and low read latency. However, they suffer from high write energy and limited write endurance. The write energy of PCM and RRAM is several times more than that of DRAM. Besides, the endurances of PCM and RRAM are 10^8 and 10^{10}, respectively [1]–[3], which are several orders of magnitude fewer than DRAM (10^{16}). After enduring limited number of bit flips (i.e., the write processes of $1 \rightarrow 0$ and $0 \rightarrow 1$ for NVM cells), cells will be worn out, and the NVM-based main memory system will fail.

If we can reduce the bit flips of NVMs, the lifetime of NVMs will be improved. Some existing works [4]–[8] propose to reduce the bit flips through data encoding. Flip-N-Write [6] encodes 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 will be set to 1. If we give fewer data bits one tag bit, Flip-N-Write can reduce more bit flips. For example, Flip-N-Write reduces the bit flips by 25% with 1 tag bit for every 2 data bits, while the reduction is decreased to 14.6% with 1 tag bit for every 16 data bits. However, the tag bits will incur significant capacity overhead if we give fewer data bits one tag bit. The capacity overhead is 50% when we give 2 data bits 1 tag bit. To avoid high capacity overhead, every 16 or 32 data bits share 1 tag bit in general. The restricted capacity overhead limits the efficiency of Flip-N-Write. Similarly, the efficiency of other encoding schemes is also limited by the capacity overhead. FlipMin [4] reduces the bit flips by 31.2% with 100% capacity overhead. The decrease of bit flips will drop to 24.5% if the capacity overhead is 12.5%. Although these methods can reduce the bit flips, the capacity overhead cannot be ignored. The capacity overhead will increase to the unacceptable degree when we want to reduce significant bit flips with data encoding.

This article aims to reduce the bit flips with negligible capacity overhead. Data encoding methods [4], [6] consume additional space to store the tag bits. Data compression techniques [9], [10] can reduce the size of the data to store and save space. The two different techniques can be combined. We propose to exploit the space saved by data compression techniques to store the tag bits of data encoding methods. Data compression techniques, e.g., frequent pattern compression (FPC) [9], can compress the 64-bit word to only 32-bit data. The remaining 32 bits are saved and can be used to store the tag bits of the data encoding methods. The space used to store the tag bits is offered by the compression techniques, and therefore data encoding methods can work with

Manuscript received April 3, 2019; revised July 16, 2019, September 24, 2019, and November 27, 2019; accepted December 7, 2019. Date of publication December 24, 2019; date of current version September 18, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61821003, Grant 61832007, Grant 61772222, Grant U1705261, and Grant 61772212, in part by the National High Technology Research and Development Program (863 Program) under Grant 2015AA015301, in part by the Shenzhen Research Funding of Science and Technology under Grant JCYJ20170307172447622, and in part by the Key Laboratory of Information Storage System, Ministry of Education, China. The preliminary manuscript was published in the proceedings of Design Automation and Test in Europe (DATE), 2018. This article was recommended by Associate Editor P. R. Panda. *(Corresponding author: Wei Tong.)*

Dan Feng, Jie Xu, Yu Hua, Wei Tong, Jingning Liu, and Chunyan Li are with the Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System, Engineering Research Center of Data Storage Systems and Technology, Ministry of Education of China, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: dfeng@hust.edu.cn; xujie_dsal@hust.edu.cn; csyhua@hust.edu.cn; tongwei@hust.edu.cn; jnliu@hust.edu.cn; lichunyan@hust.edu.cn).

Yiran Chen is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (yiran.chen@duke.edu).

Digital Object Identifier 10.1109/TCAD.2019.2962127

negligible capacity overhead. Moreover, we observe that the size of the space saved by each compressed cache line is different. Some highly compressed cache lines can offer more capacity, while slightly compressed cache lines can offer less capacity. Besides, different encoding methods have different tradeoffs between capacity and effect. Large capacity overhead can be used to reduce more bit flips. To fully exploit the saved space for reducing bit flips, we dynamically select the encoding method according to the saved space size. For example, we use the encoding method with high capacity overhead for those highly compressed cache lines.

Data compression techniques, such as FPC [9] and base–delta–immediate (BDI) [10], are frequently used in memory compression. Compared with FPC, BDI has low compression coverage (the percentage of compressible cache lines) but high compression ratio (*original size*/*compressed size*). To take the advantages of both compression techniques, we propose to compress the cache line with both FPC and BDI. After compression, we compare the saved space sizes and select the compression technique which has higher compression ratio for each cache line. Since two different compression techniques are used, more space is provided for data encoding, and encoding methods can reduce more bit flips.

Although two compression techniques are used, the compressible data patterns are still limited. Some irregular data are uncompressible. No space is offered by the uncompressible cache line, and the encoding method cannot reduce the bit flips of the uncompressed cache line. If most cache lines are uncompressible in an application, the write energy reduction will be very small. To ensure the write energy reduction of these uncompressible cache lines, we use Flip-N-Write with low capacity overhead. We give every 32-bit data one tag bit. The capacity overhead of Flip-N-Write is only 3.1%. We have the following contributions.

1) To reduce the bit flips while consuming negligible capacity, we propose to exploit the space saved by compression to store the tag bits of data encoding methods.
2) We observe that the space saved by compression varies. To fully exploit the saved space for reducing bit flips, we select the best-performing encoding method according to the size of the space saved by compression.
3) To improve the compression coverage and compression ratio and provide more space for data encoding, we propose to select the compression algorithm which has smaller compressed cache line size from FPC and BDI.
4) To ensure the bit flip reduction of uncompressible cache lines, we propose to combine our technique with Flip-N-Write.
5) The experimental results show that our scheme reduces the bit flips by 32.5%, decreases the energy consumption by 22.6% and improves the lifetime by 69.9% with 3.5% capacity overhead.

The remainder of this article 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, have the advantages of high density, nonvolatility, and fast read speed. They are considered as promising replacement candidates of the traditional DRAM technology. However, they also suffer from high write energy and limited write endurance.

PCM exploits phase change material such as $Ge_2Sb_2Te_2$ (GST) to store digital bits. When the GST is in crystalline state, the resistance of PCM cell is low, and this low-resistance state (LRS) represents the 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 10^6–10^9 bit flips [11]. The endurance of PCM is several order of magnitudes fewer than DRAM (10^{16}). Besides, PCM suffers from high write energy. The write energy of PCM is about 20 pJ/bit [12], which is twice more than DRAM. A RRAM cell consists of a top electrode and a bottom electrode, and a metal–oxide layer (HfO_2, Ta_2O_5, etc.) [13] 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 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} bit flips [3], [14] 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 and improve lifetime of NVMs. We divide the existing works into the following two categories.

1) Reducing Bit Flips With Data Encoding: Encoding methods map the new data bits into the vector with fewer bit flips. Flip-N-Write [6] divides the cache line into several N-bit data. Each N-bit data are given one tag bit. The new data are flipped to reduce the bit flips. CAFO [5] models the cache line as a number of $n \times m$ matrices and uses Flip-N-Write in both rows and columns to minimize the bit flips. Captopril [8] observes that some specific locations (i.e., hot locations) of the cache lines endure the most of the bit flips and extends Flip-N-Write to reduce bit flips in hot locations. Different from Flip-N-Write, FlipMin [4] uses coset code to generate vectors. FlipMin maps each data chunk into a set of vectors (i.e., coset). The vector that results in the minimum bit flips is selected as the encoded new data. Pseudo-Random [7] maps the data bits into highly random data vectors based on the observation that increasing the randomness of the elements in the coset can decrease the bit flips. Min-Shift [15] proposes to minimize the bit flips by shifting and flipping the bits. MFNW [16] extends Flip-N-Write to multilevel cell (MLC)/triple-level cell (TLC) NVMs by minimizing the cell Hamming distance and energy Hamming

TABLE I
DATA PATTERNS OF 64-BIT FPC [9], [28]

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

distance. ES [17] extends the methods for single-level cell (SLC) to MLC magnetic memories through encoding the hard bits and soft bits separately for fewer state transitions. Wang *et al.* [18] observed that the write energy of MLC PCM is significantly dependent on the cell states. They proposed to reduce the write energy by mapping the frequent data patterns to the low-power states. OSSR [19] optimizes static state remapping based on profiling. OISR [20] reduces the write of the intermediate states (i.e., "01" and "10") through state remapping. EARO [20] reduces the hard-bit write of MLC spin-transfer torque magnetic RAM (STT-RAM) through state remapping. Data encoding methods reduce the bit flips at the cost of capacity overhead. To gain significant bit flip reduction, data encoding techniques consume unacceptable capacity overhead.

2) Data Compression With Data Encoding: Some other works propose to combine the data encoding with data compression. Dgien *et al.* [21] proposed to compress the data bits before write operations. The compression–decompression engine (CDE) is implemented in the NVM module controller. When the NVM module receives a write access, CDE attempts to compress the cache line. During the read access, CDE decompresses the compressed cache line first. AFNW [22] extends Flip-N-Write by adapting the tag bits to the compressed data bits. Since the compressed cache line size is much smaller than the original size, AFNW can have a fine-grained encoding. In CDE [21] and AFNW [22], the saved space is wasted. DIN [23] compresses the cache line and uses the saved space to mitigate the write disturbance. DIN encodes n-bit data using m-bit code, where $m > n$, to reduce the write disturbance when the cache line is compressible. Jadidi *et al.* [24] exploited the space saved by compression for hard-error tolerance and wear leveling. COEF [25] exploits the space saved by compression to store the tag bits of the data encoding methods. An efficient encoding method is selected according to the saved space size. COEF is unable to reduce the write energy when the cache line is uncompressible. DFPC [26] exploits the distributions of 4-bit 0x0 to dynamically recognize and extend the compressible data patterns. Different from them, this article combines different data encoding methods with multiple data compression algorithms to reduce the bit flips of NVMs. For TLC NVMs, CRADE [27] and CompEx++ [28] integrate data compression with expansion coding to reduce write energy and latency. CompEx++ and CRADE first compress the cache line, and then expand the compressed cache line. During the encoding, it is ensured that the expanded cache line size does not exceed the cache line size. WLCRC [29] proposes a high coverage compression scheme and uses the saved space to store the tag bits of the restricted coset code. WIPE [30] proposes to use static and dynamic profiling to find the frequent data patterns and encodes them into a fewer number of bits. Some other works [31]–[34] propose to dynamically configure the MLC as SLC, tri-state cell, or MLC according to the size of the space saved by compression.

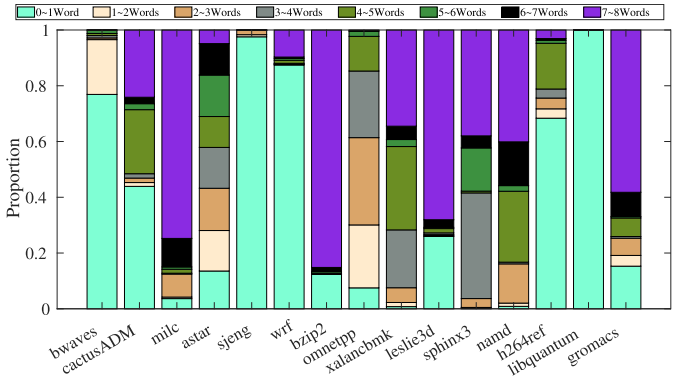


Fig. 1. Distribution of the compressed cache line size.

III. MOTIVATION

A. Size of the Space Saved by Compression Is Various

Various compression techniques have been proposed. We take FPC [9] as an example in this section to evaluate the saved space size of compression algorithm. The original FPC [9] compresses the data at the granularity of 32-bit word. 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 word in this article. Table I lists the data patterns that 64-bit FPC can compress [9], [28]. A 64-bit word can be compressed to 0, 8, 16, or 32 bits. The compressed word size may be different. For a cache line which consists of eight words, the compressed size of each word is different, and the size of the compressed cache line is also different. The total number of bits in the compressed cache line may range from 0 to 512. We analyze the distribution of the compressed cache line sizes through experiments. The detailed system configuration is shown in Table IV. Fig. 1 shows the compressed cache line size distribution for 15 benchmarks selected from SPEC CPU 2006 [35]. In the bwaves benchmark, the sizes of about

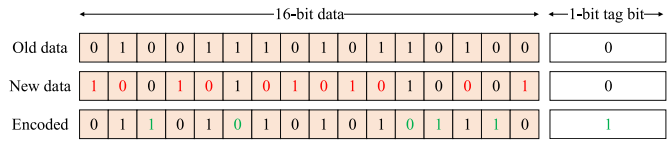


Fig. 2. Example of Flip-N-Write.

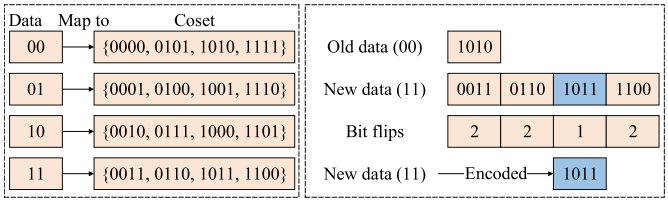


Fig. 3. Example of FlipMin.

half of the compressed cache lines are between 0 and 1 word. The saved space sizes are between 7 and 8 words, and these cache lines are highly compressed and can offer large space for encoding. The compressed sizes of about 1% cache lines in the bwaves benchmark are between six words and eight words (slightly compressed). For these slightly cache lines, the saved space is limited. Among the 15 benchmarks, a large amount of the cache lines are highly compressed in the bwaves, sjeng, wrf, h264ref, and libquantum benchmarks, while most cache lines are slightly compressed in the milc, bzip2, leslie3d, and gromacs benchmarks. We conclude that the saved space sizes are different for different data patterns. For the highly compressed cache lines, more space is saved and offered to data encoding. In contrast, the slightly compressed cache lines have less space for data encoding.

B. Tradeoffs Exist Between Effect and Capacity Overhead in Encoding Methods

Most of the existing encoding schemes are designed based on Flip-N-Write [6] or FlipMin [4]. We take Flip-N-Write [6] and FlipMin [4] as examples to discuss the tradeoffs between effect and capacity overhead.

1) Flip-N-Write: Flip-N-Write reduces the bit flips by flipping the data to be written. In Flip-N-Write [6], a cache line 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 data bits need to be flipped, and the tag bit will be set to 1. Fig. 2 illustrates an example of Flip-N-Write. One tag bit is assigned to every 16-bit data. Writing the new data incurs 11 bit flips (indicated in red color), which exceeds $(16+1)/2$. Therefore, the new data are flipped, and only six bit flips (indicated in green color) are required. The bit flip of the tag bit is also included in the 6 bit flips. The reduction of bit flips will decrease if N increases. When N equals 2, the effect of Flip-N-Write is the best, and Flip-N-Write can reduce 25% more bit flips than DCW [36]. The reduction drops to 14.6% when N equals 16. However, the capacity overhead of Flip-N-Write will increase significantly if N decreases. The capacity overhead is 50% when N equals 2, and the overhead drops to 6.25% when 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] uses the coset code to minimize the bit flips. Each N-bit data is mapped to a set of vectors, and FlipMin chooses the vector that has the minimum bit flips. Fig. 3 shows an example of FlipMin. Every 2-bit data are mapped into four vectors. The set of the four vectors is called "coset." When writing the new data "11," we select the vector "1011" from the coset because the vector 1011 incurs only 1 bit flip. The bit flip 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% when capacity overhead is 12.5%. The latency overheads of encoding and decoding in FlipMin vary significantly. The coset code with 100% capacity overhead (RM(1, 3)) incurs 4.09-ns encoding latency and 0.38-ns decoding latency [4], while the coset code with 12.5% capacity overhead (RM(1, 7)) incurs 12.86-ns encoding latency and 0.59-ns decoding latency [4]. RM(1, 3) and RM(1, 7) belong to the Reed–Muller code. The Reed–Muller code can be used to generate the coset. The general form of the Reed–Muller code is RM(r, m). RM(1, 3) is used to map each 4-bit data into a set of 16 vectors. Each vector has 8-bit data.

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

		Encoding Latency	Decoding Latency	Bit flip 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%

Besides, the tradeoffs exist between different encoding methods, e.g., Flip-N-Wite and FlipMin. The encoding latency, decoding latency, bit flip reduction, and capacity overhead comparisons of Flip-N-Write and FlipMin are shown in Table II. In Flip-N-Write, N can be set to any values, not just 2, 4, 8, or 16. Flip-N-Write can use at most 50% additional capacity as tag bits, while FlipMin can use more than 100% additional capacity. With the same capacity overhead, FlipMin can reduce more bit flips than Flip-N-Write. FlipMin can reduce 6.5% more bit flips than Flip-N-Write with the same 12.5% capacity overhead. In the aspect of latency, the encoding/decoding process of Flip-N-Write is very simple and fast. The encoding latency of FlipMin with 12.5% capacity overhead is about 12.86 ns, which is ten times more than Flip-N-Write.

IV. DESIGN AND IMPLEMENTATION

A. Design

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. Moreover,

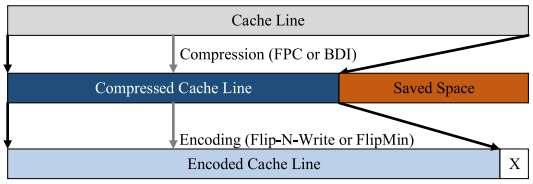


Fig. 4. Overview of our scheme.

the sizes of the space saved by compression are various for different patterns. Encoding methods with different granularities also consume different capacity. We also propose to select an encoding method according to the saved space size. For these highly compressed cache lines, more space is provided to the encoding schemes, and the encoding schemes can have a fine-grained granularity and high efficiency. For the slightly compressed cache lines, the saved space can also be used by coarse-grained encoding schemes.

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. Our scheme works as Fig. 4 describes. For each cache line, we first use the compression technique to compress the cache line. Different compression techniques, such as BDI and FPC, can be used. If the cache line is uncompressible, we will not encode it because no extra space is provided. If the cache line is compressible, we will compress it and use the saved space to store the tag bits. Since different cache lines have different compressed sizes and can provide various saved space, a suitable encoding method is chosen according to the saved space size for each cache line. The encoding schemes use the saved space to store the tag bits, and therefore the encoding scheme does not consume additional capacity. To efficiently reduce the bit flips, our detailed design has the following four optimizations.

2) Prefix Consolidation: Compression techniques use prefixes to indicate the data patterns. We take FPC compression technique as an example in this section. In the original FPC compression scheme, each compressed word uses a 3-bit prefix to indicate the data pattern or compressibility. A cache line (eight words) requires 24-bit prefixes. The prefixes consume additional 24-bit space. We propose prefix consolidation to reduce the storage overhead and decompression latency of prefixes in compression technique. To reduce the capacity overhead of the prefixes, we modify the organization of prefixes and the data. We use a tag bit (compression tag) to indicate whether the cache line is compressible or not. If there is at least one compressible word, the cache line will be considered as compressible and the tag bit will be set. Otherwise, the cache line is uncompressible, and the tag bit is reset. For the compressible cache line, the saved space size is 32 bits at least. For each word of the compressible cache line, we still use a 3-bit prefix to indicate the data pattern. The total number of bits of prefixes in compressible cache line is 24, which is smaller than 32. Therefore, the size of the compressible cache line with 24-bit prefix will not exceed 512. For the uncompressible cache lines, we do not use prefixes and each word is stored without any modification. Through prefix consolidation, the capacity overhead of compression is reduced to only one bit.

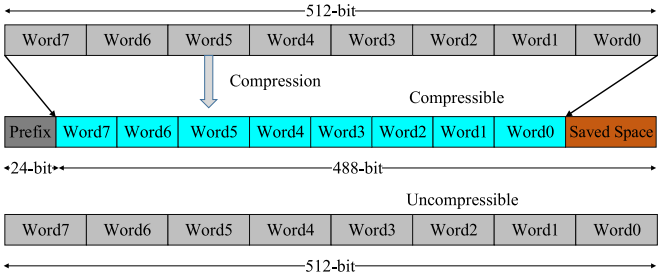


Fig. 5. Organization of prefixes and eight compressed words for the compressible cache line.

Prior works [22], [28] propose 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 causes accumulated decompression latency. We propose to place the prefixes together, as shown in Fig. 5. The location of each compressed word can be located by the first 24-bit prefixes with parallel decompression.

3) Selective Encoding: To fully exploit the saved space for reducing the bit flips, we propose selective encoding. Flip-N-Write and FlipMin have some variations. We use the encoding methods with low encoding overheads to avoid the IPC performance degradation. All the variations of Flip-N-Write have low encoding/decoding overheads, and we can select any encoding scheme from the variations of Flip-N-Write. FlipMin with RM(1, 3) has moderate encoding/decoding overhead. Compared with Flip-N-Write and FlipMin with RM(1, 3), FlipMin with RM(1, 7) consumes significant latency and energy overheads. The encoding latency of RM(1, 7) is 12.86 ns, which is 8.6% of the write latency of PCM. The encoding energy is 63.4 pJ [4], which is three times the write energy of a PCM cell. Therefore, we do not use FlipMin with RM(1, 7). The encoding method candidates we use in this article are FlipMin with RM(1, 3) and all the variations of Flip-N-Write. In the following pages, FlipMin with RM(1, 3) is abbreviated to FlipMin, since FlipMin with RM(1, 7) is not used.

Flip-N-Write and FlipMin have tradeoffs in capacity overhead and effect. The tag bits of Flip-N-Write are at most 50% of the data bits, while FlipMin consumes 100% capacity overhead. When selecting from Flip-N-Write and FlipMin, we choose Flip-N-Write when the saved space is small. Flip-N-Write cannot fully use the saved space if the saved space is more than half of the data bits. We use FlipMin when the saved space size is larger than the data size. 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 each cache line, S stands for the saved space size of the eight words, and D is the total number of bits in the eight compressed words. When 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. We give every D/S data bits one tag bit. For example, if only one of the eight words is compressible, and the data

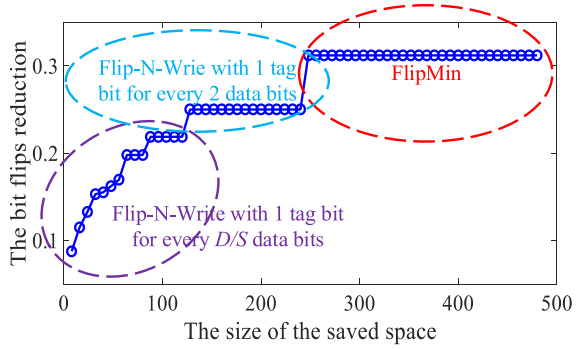


Fig. 6. Relationship between the saved space size and the bit flip reduction.

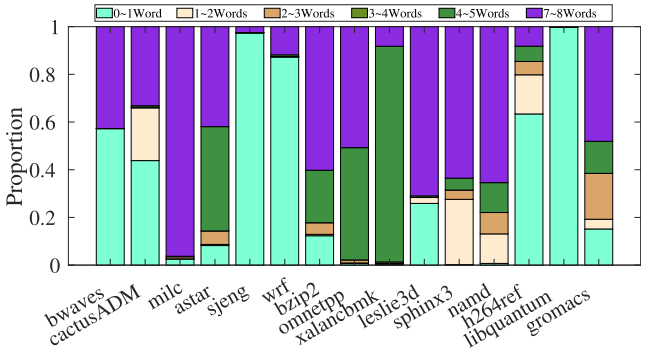


Fig. 7. Distribution of the sizes of the cache line compressed by BDI.

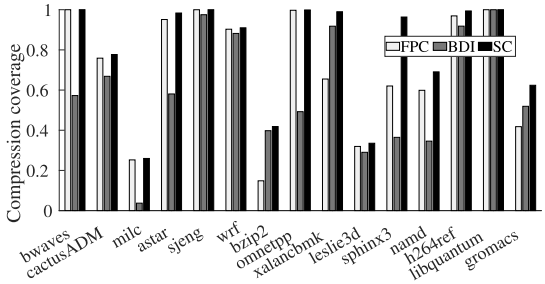


Fig. 8. Percentage of compressible cache lines.

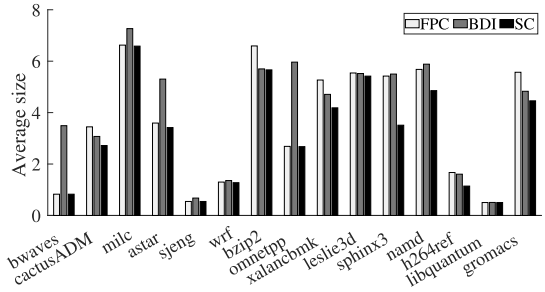


Fig. 9. Distribution of the average compressed cache line sizes.

pattern is "011," the saved space size (S) equals "8" (32-24). D equals "480" $(32 + 64 + 64 + 64 + 64 + 64 + 64 + 64)$. The result of D/S is 60. Therefore, we give every 60 data bits one tag bit. If S/D is between 50% and 100%, the saved space size will not be enough for FlipMin. We use the most fine-grained Flip-N-Write, i.e., give every 2 data bits 1 tag bit in Flip-N-Write. When 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. 6. When the saved space size (S) is between 8 and 162, D/S is between 60 and 2. We give every D/S data bits one tag bit. When S is between 163 and 243, we give every 2 data bits one tag bit. When S is between 244 and 488, we use the FlipMin with 100% capacity overhead. The relationship between the encoding granularity and theoretical efficiency of bit flip reduction is shown in (1). In (1), R represents the bit flip reduction, and N equals the integer portion part of D/S

$$R = \begin{cases} 1 - \sum_{i=0}^{N/2} i \times \binom{N+1}{i} / \left(N \times 2^{N-1}\right) & 8 \le S \le 162 \\ 25\% & 163 \le S \le 243 \\ 31.2\% & 244 \le S \le 488. \end{cases} \quad (1)$$

4) Selective Compression: Compression techniques are designed based on the data features. FPC is based on the observation that some frequent data patterns can be represented by a fewer number of bits. Recently proposed BDI compression technique is designed based on that the differences between data values in the same cache line are small. BDI uses a *base* and several *delta*s to represent the original cache line. BDI is a cache line level compression algorithm which can compress eight different data patterns. The size distribution of the cache lines compressed by BDI is shown in Fig. 7. The sizes of a large number of cache lines are on either sides, i.e., the sizes of the majority of the compressed cache lines are either smaller than 1 word or larger than 7 words. In the BDI algorithm, no compressed cache line has the size between 5 and 7 words.

FPC and BDI have different compression characteristics. FPC compresses the data at the granularity of word. There is a high probability that at least one of the eight words is compressible. The compression coverage (the percentage of compressible cache lines) of FPC may be high. But the prefixes of the eight compressed words will occupy 24-bit capacity. The minimum compressed cache line size (with the prefixes) is 24. On the contrary, BDI compresses the data at the granularity of a cache line. The data pattern of a cache line is indicated by a 3-bit prefix. Therefore, the compression ratio (*original size*/*compressed size*) of BDI is high. The data regularity requirement of BDI is stricter than FPC. All the eight words of the compressible cache line must meet the same condition. Therefore, the compression coverage of BDI may be lower than FPC. If we use both compression schemes to compress data, the compression ratio and compression coverage will be higher, and more space will be provided to data encoding. We propose selective compression (SC) which improves the compression ratio and compression coverage by selecting the efficient compression algorithm from FPC and BDI.

Since two different compression techniques are used, SC can take the advantages of high compression coverage and high compression ratio. We also analyze the compression ratio and compression coverage of the three compression schemes through experiments. Figs. 8 and 9 show the compression coverages and average compressed cache line sizes of the FPC, BDI, and SC. The experimental setup is given in Table IV. The compression coverage of FPC is higher than BDI in the benchmarks of bwaves, cactusADM, milc, astar, omnetpp, sphinx3,

Algorithm 1 Selective Compression and Encoding

Require: *OldLine*, *NewLine*: the old and new cache lines
Require: *P*, *L*: the sizes of the prefix and cache line
Ensure: *LE*: the encoded new cache line
1: /* Selective compression */
2: $LF \leftarrow FPCCompression(NewLine)$ ▷ LF is the line compressed by FPC
3: $LB \leftarrow BDICompression(NewLine)$ ▷ LB is the line compressed by BDI
4: $LC \leftarrow size(LF) < size(LB)?LF : LB$ ▷ LC is the line compressed by SC
5: /* Selective encoding */
6: $D \leftarrow size(LC)$ ▷ D is the size of the compressed line
7: $S \leftarrow L - P - D$ ▷ S is the saved space size
8: **if** $S == 0$ **then**
9: $LE \leftarrow LC$
10: **else if** $S < D$ **then**
11: $LE \leftarrow Flip - N - WriteEncoding(OldLine, LC)$;
12: **else if** $S >= D$ **then**
13: $LE \leftarrow FlipMinEncoding(OldLine, LC)$
14: **end if**

namd, and h264ref. The average compression coverages of the 15 benchmarks are 70.6%, 59.8%, and 79.6% in FPC, BDI, and SC, respectively. SC always has higher compression coverage than FPC and BDI due to selecting the smaller compressed cache line size. The average compressed cache line size of BDI is smaller than FPC in the benchmarks of cactusADM, bzip2, xalancbmk, and gromacs. The two compression schemes have similar compressed sizes in the benchmarks of sjeng, wrf, leslie3d, sphinx3, namd, h264ref, and libquantum. The average compressed cache line sizes are 3.7 words, 4.1 words, and 3.2 words in FPC, BDI, and SC, respectively. SC also has smaller compressed cache line size than FPC and BDI in all the benchmarks.

When selecting from the two compression techniques, we need to calculate the number of bit flips of the compression techniques, compare their values, and find the smaller one. However, calculating the number of the total bit flips of the cache line is very slow and incurs high latency overhead. The average number of bit flips of writing the compressed cache line is half of the compressed cache line size. To reduce the overhead of the selection operation, we use the compressed cache line size to approximately measure the number of bit flips. We select the compression technique which has smaller compressed cache line size. There is no need to calculate the compressed cache line size, because the size can be obtained directly by the prefixes. We only need to compare the existing compressed cache line sizes and find the smaller one. The selective compression and encoding algorithms are shown in Algorithm 1.

5) Combined With Flip-N-Write: Although selective compression can improve the compression coverage and reduce the compressed cache line size, some random and irregular data patterns are still uncompressible. If most of the access patterns in the application are uncompressible, the proposed technique will not reduce the write energy. As shown in Figs. 8 and 9, the milc and bzip2 benchmarks have low compression coverages and high compressed cache line sizes. The saved space sizes are small in the two benchmarks. To ensure the reduction

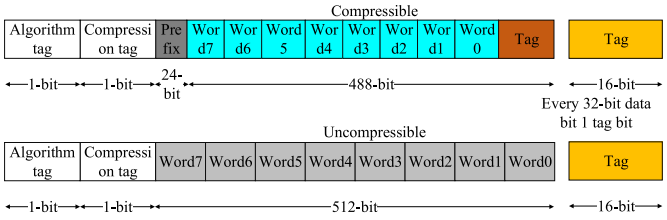


Fig. 10. Combination of Flip-N-Write with compression encoding.

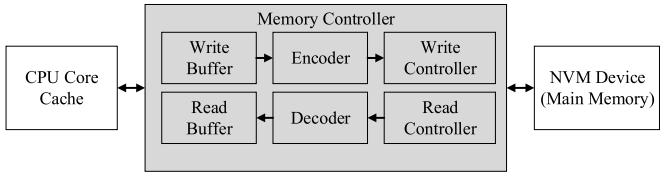


Fig. 11. System architecture.

of write energy for these benchmarks, we propose to combine Flip-N-Write with the compression-encoding scheme. We give every 32-bit data one tag bit to ensure low capacity overhead. The capacity overhead of Flip-N-Write is only 3.1%. If we use Flip-N-Write before compression, the data characteristics will be destroyed. Therefore, the Flip-N-Write works after the compression and data encoding scheme. Fig. 10 shows the combination of compression encoding with Flip-N-Write. For the compressible cache lines, we apply compression and encoding first. Then, we use Flip-N-Write to encode the "encoded cache line." If the cache line is uncompressible, we will directly apply Flip-N-Write to the cache line. Although the Flip-N-Write in this section incurs additional capacity overhead, it ensures the bit flip reduction for all the cache lines.

B. Implementation

NVM is used as main memory in this article. When there is a write access to the NVM device, we use the Encoder to encode the cache line first. When there is a read access, we read the stored data and decode the data. The implementation of our design includes Encoder and Decoder, which are on the write path and read path, respectively. The system architecture is shown in Fig. 11. The Encoder and Decoder of our scheme are implemented in the main memory controller.

1) Encoder: The Encoder attempts to compress the cache line and use Flip-N-Write or FlipMin to encode the compressed data. The Encoder consists of two parts, i.e., selective compression and selective encoding, as shown in Fig. 12. When the memory controller receives a write request, the cache line is sent to the FPC compression logic and BDI compression logic to attempt data compression. For the FPC algorithm, each of the eight words is compared with the data patterns illustrated in Table I. If none of the eight words is compressible, this cache line will be uncompressible by FPC. For the BDI algorithm, the cache line is matched with the *base* and several *deltas*. If the match is successful, the cache line will be compressible by BDI. If the cache line is either compressible by FPC or BDI, the cache line will be marked

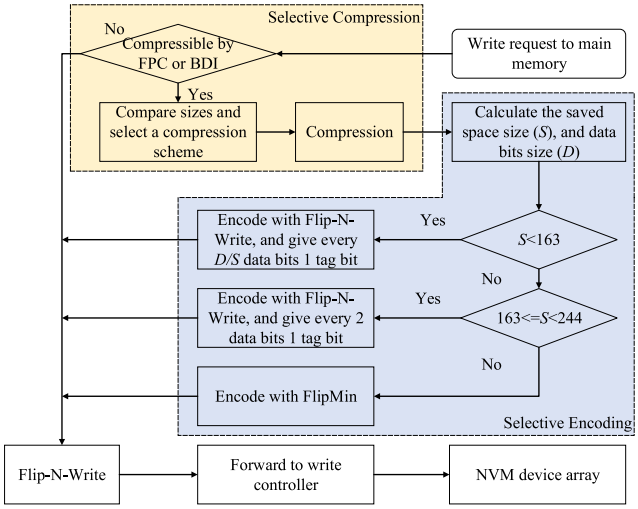


Fig. 12. Process of the Encoder.

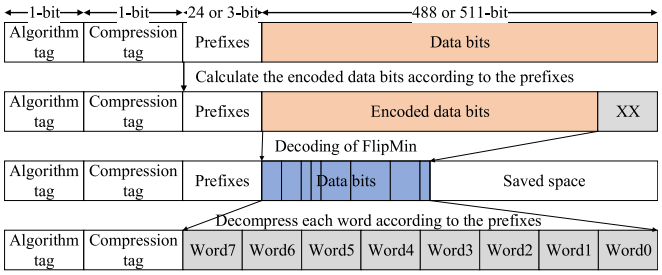


Fig. 13. Process of the Decoder.

as compressible by setting the compression tag. Then, we compare the compressed cache line sizes of FPC and BDI and select the compression algorithm which has smaller compressed cache line size. If FPC is applied, the algorithm tag will be set. Otherwise, the algorithm tag is reset. If the cache line is uncompressible by both FPC and BDI, the compression tag will be reset.

For the compressible cache line, the cache line is compressed by either FPC or BDI, and then the saved space size (S) and compressed data bits size (D) are calculated. Different encoding methods are applied to the compressed cache line based on the value of S. If S is smaller than 163, we will choose Flip-N-Write and give every D/S data bits 1 tag bit. If S is between 163 and 244, we will choose Flip-N-Write with 1 tag bit for every 2 data bits. If S is greater than 244, we will select the FlipMin with 100% capacity overhead. After the selective compression/encoding, all the cache lines are encoded by Flip-N-Write before they are forwarded to the write controller.

2) Decoder: Since the cache line 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, we decode the cache line with Flip-N-Write. The compression tag is used to determine that whether the cache line is compressed or not. If the cache line is not compressed, it will be sent to the read buffer directly. If the cache line is compressed, the algorithm tag will be used to identify the compression algorithm. If the value of algorithm tag is 1, the cache line will be decompressed by FPC. For the compressed cache line, 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. Assuming that the prior compression and encoding methods are FPC and FlipMin, the Decoder works as Fig. 13 illustrates.

TABLE III
OVERHEAD COMPARISON

		Energy	Latency
BDI	Encoding	3.9pJ	2ns
	Decoding	-	-
FPC	Encoding	2.1pJ	2ns
	Decoding	1.2pJ	1ns
Selecting Compression	Encoding	0.31pJ	0.43ns
	Decoding	-	-
Flip-N-Write	Encoding	6.1pJ	0.54ns
	Decoding	-	-
FlipMin	Encoding	8.5pJ	4.09ns
	Decoding	0.3pJ	0.38ns
Selecting Encoding	Encoding	1.7pJ	1.52ns
	Decoding	-	-

C. Overhead

The Encoder and Decoder incur additional latency and energy overheads. The implementation consists of three parts, i.e., compression, encoding, and selecting encoding and compression methods. The average encoding energy is about 11 pJ, which is about half of the write energy of a PCM cell. The encoding latency is between 2 and 8 ns. The average encoding latency is 5 ns, which is 3.3% of the write latency of PCM. The overhead is small compared with the write energy/latency of PCM. Besides, the selective compression and encoding schemes will incur latency/energy overhead only if the cache line is compressible. The detailed latency and energy overheads of compression, encoding, and selection are shown in Table III. The symbol "−" means that the value is negligible.

1) Compression: The compression schemes include FPC and BDI. The hardware, energy, and latency overheads of FPC have been evaluated in prior works [9], [23], [31]. The hardware overhead is similar to 16K PCM cells. The encoding/decoding energy is 2.1/1.2 pJ [31]. The latencies of compression and decompression are estimated to be 2 and 1 ns [28]. BDI [10] has low hardware implementation complexity compared with FPC. The compression latency is about 2 ns. Besides, the compression/decompression of BDI can work in parallel with FPC. Therefore, BDI does not incur additional compression/decompression latency overhead.

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 0.3/8.4 pJ [4]. The latency and energy overheads of FlipMin are evaluated with Synopsys Design Compiler and IC Compiler based on the Nangate 45-nm semi-custom cell library [4].

TABLE IV
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, 20pJ/bit

3) Selecting Compression and Encoding: The proposed selective compression scheme needs to calculate and compare the compressed cache line sizes. In BDI, the compressed cache line size can be obtained directly according to the prefix. In FPC, the compressed cache line size is the sum of the eight compressed words. We need to add the sizes of the compressed words. The estimated energy and latency overheads of selecting compression algorithms are 0.31 pJ and 0.43 ns. The selective encoding scheme needs to calculate the saved space size and select an encoding schemes according to the saved space size. The latency of selecting encoding schemes is 1.52 ns, and the energy overhead is 1.7 pJ. We get the overhead values by using Synopsys Design Compiler to synthesize the logic in 130-nm technology and scaling the results down to 22-nm technology node. During the decoding process, the compression algorithm and encoding scheme are identified by the value of algorithm tag and compression tag. Therefore, the decoding of selecting compression and encoding does not incur latency or energy overhead.

V. EXPERIMENTAL SETUP

We use Gem5 [37] to evaluate our schemes, and the main memory model is based on NVMain [38]. 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 IV. The system is based on a four-core processor. Fifteen benchmarks are used in our experiment. All these benchmarks are selected from SPEC CPU 2006 [35].

The evaluation section is divided into two parts. In the first part, we evaluate the effects of different combinations of encoding schemes and compression techniques. In the second part, we compare our schemes with existing works.

A. Combining Encoding With Compression

In this section, we evaluate the effects of different combinations of encoding schemes and compression algorithms to show the tradeoffs in encoding and compression. All the schemes use DCW to reduce the redundant bit flips. The proposed designs are evaluated in terms of bit flips, energy, and lifetime. All the experimental results are normalized to DCW [36]. The following six different schemes are evaluated.

1) *FPC + FNW:* The space saved by FPC is exploited to store tag bits of Flip-N-Write. Each cache line is compressed by FPC first. Then, Flip-N-Write is used to encode the compressed cache line according to the saved space size.
2) *FPC + FlipMin:* The space saved by FPC is used to store the tag bits of FlipMin. FlipMin is applied when the saved space size is larger than the compressed cache line size.
3) *BDI + FNW:* Cache lines are compressed by BDI. Flip-N-Write is used to encode the compressed cache line.
4) *BDI + FlipMin:* Cache lines are compressed by BDI. FlipMin is used to encode the compressed cache line.
5) *COEF:* FPC is used to compress the cache lines. FlipMin or Flip-N-Write is used to encode the compressed cache line according to the saved space size.
6) *SELEC:* Both FPC and BDI are used to compress the cache lines. The compressed cache line which has a smaller size is selected. Then, Flip-N-Write or FlipMin is used to encode the compressed cache line.

TABLE V
COMPARISON OF CAPACITY OVERHEAD, BIT FLIPS, ENERGY, AND LIFETIME (NORMALIZED TO DCW)

	Capacity overhead	Bit flips	Energy	Lifetime
FPC+FNW	0.2%	0.98	1.00	1.03
FPC+FlipMin	0.2%	0.91	0.95	1.14
BDI+FNW	0.2%	0.91	0.93	1.13
BDI+FlipMin	0.2%	0.96	0.96	1.06
COEF	0.2%	0.84	0.91	1.23
SELEC	0.4%	0.79	0.87	1.31

The overview of the comparison of capacity overhead, bit flips, energy, and lifetime is shown in Table V. The capacity overhead is defined as the ratio of additional space and main memory space. For example, SELEC has two tag bits (i.e., algorithm tag and compression tag) for each cache line, and therefore the capacity overhead is 0.4% (2/512). The detailed bit flips, energy, and lifetime results are shown in the following sections.

1) Bit Flips: Fig. 14 illustrates the normalized bit flips for each benchmark. The average bit flip reductions of FPC + FNW, FPC + FlipMin, BDI + FNW, BDI + FlipMin, COEF, and SELEC are 2.1%, 8.9%, 9.5%, 3.9%, 15.7%, and 20.7%, respectively. COEF and SELEC can reduce the most bit flips due to their selective encoding and selective compression. COEF reduces more bit flips than both FPC + FNW and FPC + FlipMin because COEF selects the encoding schemes from Flip-N-Write and FlipMin, and COEF can fully exploit the saved space. SELEC has fewer bit flips than COEF because SELEC uses two different compression algorithms. SELEC can compress more cache lines and provide more space for efficient data encoding. In the bwaves benchmark, FPC + FlipMin can reduce 27.8% bit flips compared with FPC + FNW. The reason is that the sizes of about 90% of the compressed cache lines are between 0 and 4 words, as shown in Fig. 1. FlipMin can significant reduce the bit flips of these cache lines. In the bwaves benchmark, BDI + FNW and BDI + FlipMin have the similar bit flips. The reason is that the size distribution of BDI compression is polarized. As shown in Fig. 7, about 60% of the compressed cache lines have 0-1 word (i.e., the 3-bit prefix of BDI), and about 40% of the cache lines are uncompressible (between 7 words and 8 words). For those highly compressed cache lines, both Flip-N-Write and FlipMin have limited effects. For those uncompressible cache

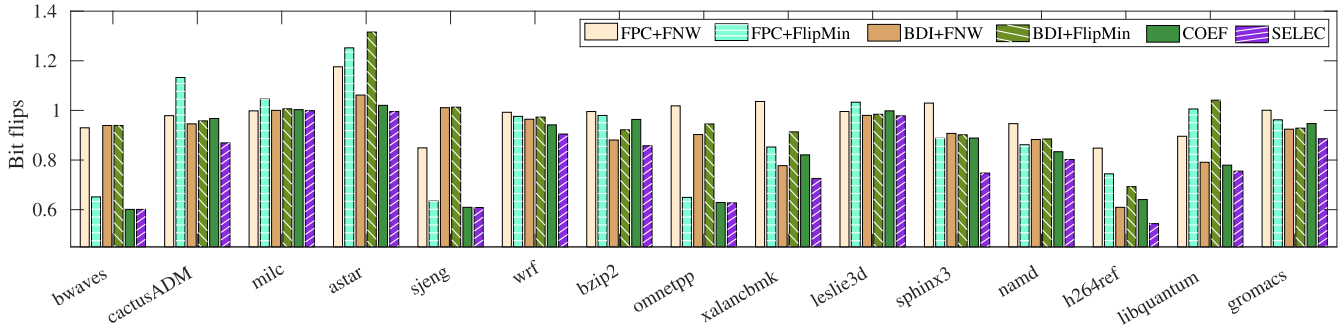


Fig. 14. Comparison of the normalized bit flips.

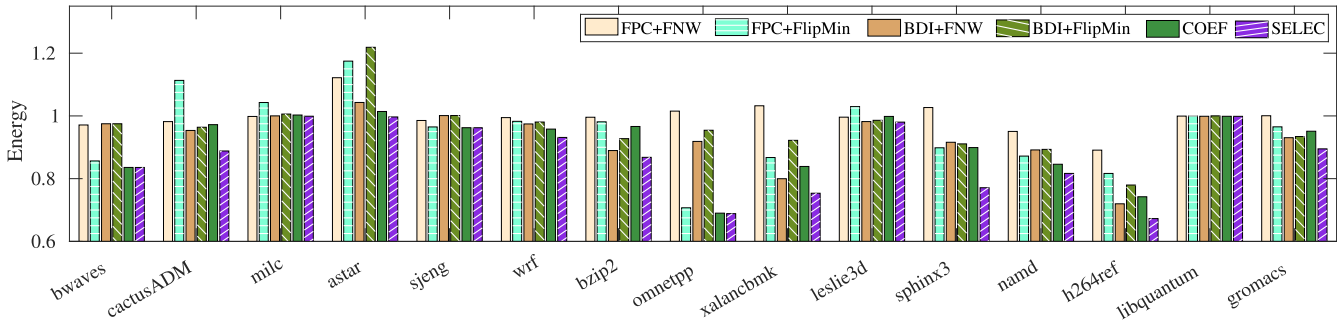


Fig. 15. Comparison of the normalized energy.

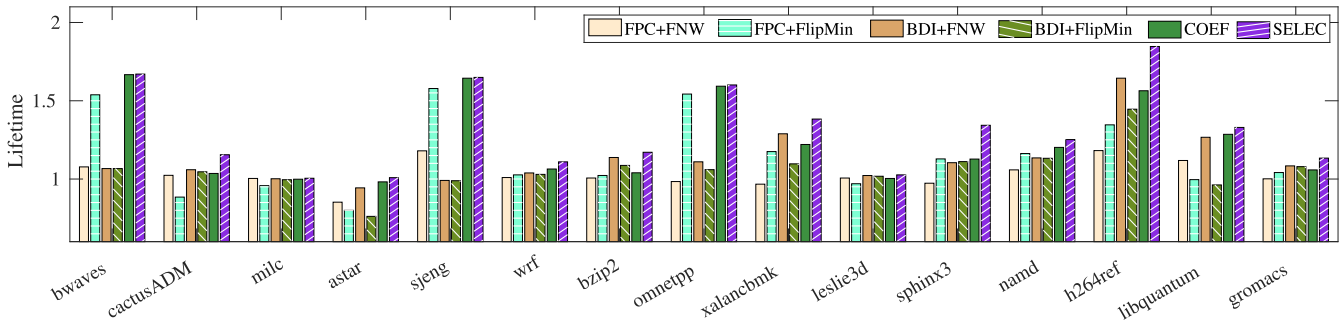


Fig. 16. Comparison of the normalized lifetime.

lines, Flip-N-Write and FlipMin cannot be applied. Therefore, BDI+FNW has the similar bit flips to BDI + FlipMin in THE bwaves benchmark. COEF and SELEC can select the encoding schemes and compression algorithms, and therefore COEF and SELEC can reduce more bit flips than all the other four schemes.

In some benchmarks, the bit flips after compression and encoding increase rather than decrease. For example, in the astar benchmark, the bit flips increase by 17%, 25%, 6%, and 31% in FPC + FNW, FPC + FlipMin, BDI + FNW, and BDI + FlipMin, respectively. The reason is twofold. On the one hand, compression algorithm could destroy the data similarity, and leads to more bit flips than DCW [24], [39]. Dirty cache line may have clean words, and DCW can eliminate the writes to those clean words. The clean words after compression are different from the original words, and compression leads to more bit flips in this case. On the other hand, FlipMin (or Flip-N-Write) uses additional tag bits and increases the size of the data to write. The bit flips increase rather than decrease under specific data patterns [40].

2) Energy: The total energy consumptions of different schemes are shown in Fig. 15. The total energy consumptions are reduced by 0.3%, 4.9%, 6.7%, 3.6%, 8.8%, and 13.0% in FPC + FNW, FPC + FlipMin, BDI + FNW, BDI + FlipMin, COEF, and SELEC. For NVMs, write energy consumption dominates the total energy consumption. Therefore, the energy reduction is nearly the same as the bit flip reduction in most benchmarks (i.e., cactusADM, milc, wrf, bzip2, xalancbmk, leslie3d, sphinx3, namd, and gromacs). For other benchmarks, the energy consumption reductions are smaller than the bit flip reductions because the write energy contributes partially to the total energy consumption.

3) Lifetime: The endurance of NVM is limited. After enduring certain number of bit flips, a cell may fail. To tolerate the wearout of cells, error correction [41] and spare lines [42]–[44] are used. When the number of errors exceeds the correction capability of ECP [41] or spare lines [42]–[44], the NVM-based main memory system will fail. For simplicity, we use the average number of bit flips of cells to approximate the lifetime. Therefore, the lifetime is inversely proportional

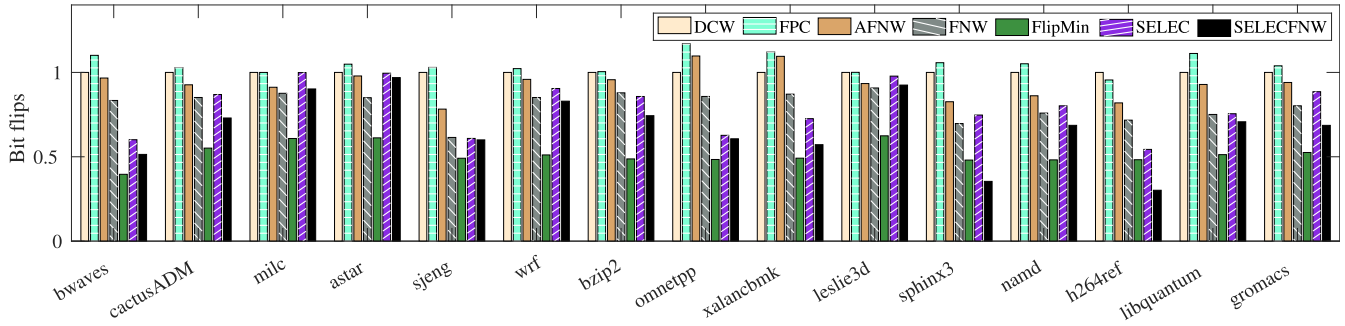


Fig. 17. Comparison of the normalized bit flips.

to the number of bit flips, as defined in (2). C is a constant value for all the schemes.

$$\text{Lifetime} = C \times \frac{\text{Total Number of Cells}}{\text{Total Number of Bit Flips}}. \quad (2)$$

Our schemes can reduce the bit flips, and the lifetime is improved. The lifetime improvements are 3.0%, 14.5%, 12.6%, 5.9%, 23.2%, and 31.2% in FPC + FNW, FPC + FlipMin, BDI + FNW, BDI + FlipMin, COEF, and SELEC, respectively, as shown in Fig 16. In the bwaves, sjeng, and omnetpp benchmarks, the lifetime improvements are significant in FPC + FlipMin. The main reason is that the high compression coverage and small compressed size lead to fewer bit flips in the three benchmarks. COEF and SELEC also have significant lifetime improvements due to their selective compression and selective encoding.

The experimental results confirm that SELEC can reduce the bit flips and improve the lifetime significantly compared with single encoding scheme or compression algorithm.

B. Compared With Prior Works

In this section, we compare our schemes with five existing works in terms of bit flips, energy, lifetime and IPC performance. All the experimental results are normalized to DCW. The following seven schemes are evaluated.

1) *DCW [36]:* The redundant bit flips are eliminated.
2) *FPC [9]:* The cache line is compressed by FPC at the granularity of 64-bit words.
3) *AFNW [22]:* The tag bits are assigned to the compressed data bits. Each 64-bit word has four tag bits in AFNW.
4) *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.
5) *FlipMin [4]:* We use the RM(1, 3) to generate the coset. Each 4-bit data are mapped into a set of 16 8-bit vectors. The vector which causes the minimum bit flips are selected.
6) *SELEC:* Both FPC and BDI are used to compress the cache lines. The compressed cache line which has a smaller size is selected. Then, Flip-N-Write or FlipMin is used to encode the compressed cache line according to the saved space size.
7) *SELECFNW:* FPC and BDI are used to compress the cache lines. Flip-N-Write and FlipMin are used to encode the compressed cache lines. Besides, the Flip-N-Write with additional 3.1% capacity overhead is used to reduce the bit flips of the encoded cache lines.

TABLE VI
COMPARISON OF CAPACITY OVERHEAD, BIT FLIPS, ENERGY, AND LIFETIME (NORMALIZED TO DCW)

	Capacity overhead	Bit flips	Energy	Lifetime
DCW	0.0%	1.00	1.00	1.00
FPC	1.56%	1.05	1.03	0.96
AFNW	6.25%	0.93	0.96	1.15
Flip-N-Write	12.5%	0.81	0.88	1.41
FlipMin	100%	0.52	0.66	3.93
SELEC	0.4%	0.79	0.87	1.31
SELECFNW	3.5%	0.68	0.77	1.70

The overview of the comparison of capacity overhead, bit flips, energy, and lifetime is shown in Table VI.

1) Bit Flips: Fig. 17 illustrates the normalized bit flips for each benchmark. Compared with DCW, the average bit flip reductions of FPC, AFNW, Flip-N-Write, FlipMin, SELEC, and SELECFNW are −5.0%, 6.8%, 19.2%, 48.4%, 20.7%, and 32.5%, respectively. FPC leads to 5% more bit flips than DCW due to destroying the data similarity [24], [39]. Besides, FPC needs 24-bit prefixes per cache line to indicate the data patterns, and the writes to prefixes result in additional bit flips. AFNW reduces the bit flips by 6.8% because it adapts the Flip-N-Write granularity to the compressed cache line size. FlipMin can reduce the most bit flips because it exploits the coset code to generate a coset of vectors and selects the vector which leads to the minimum bit flips. However, FlipMin has significant capacity overhead (100%). On average, the proposed SELEC can reduce the similar bit flips to Flip-N-Write. In the aspect of capacity overhead, SELEC has very low capacity overhead and saves 12.1% capacity. SELECFNW further reduces the bit flips by leveraging additional space to encode the cache lines with Flip-N-Write. On average, SELECFNW can reduce 11.8% more bit flips than SELEC. The bit flip reduction of SELECFNW is the highest in the sphinx3 and h264ref benchmarks. The main reason is that selective encoding and selective compression can reduce bit flips, and a low-overhead Flip-N-Write further enhances the bit flip reduction.

2) Energy: The total energy consumptions of different schemes are shown in Fig. 18. The average energy reductions of FPC, AFNW, Flip-N-Write, FlipMin, SELEC, and SELECFNW are −3.2%, 4.2%, 12.5%, 34.2%,

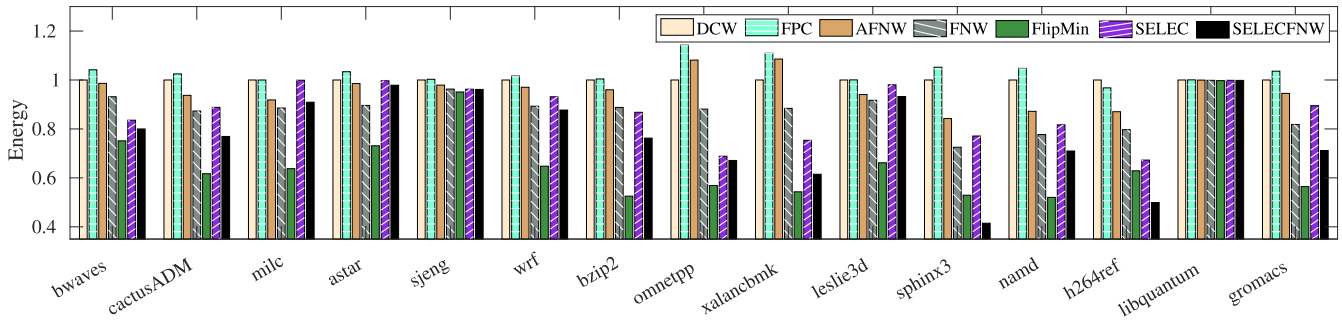


Fig. 18. Comparison of the normalized energy.

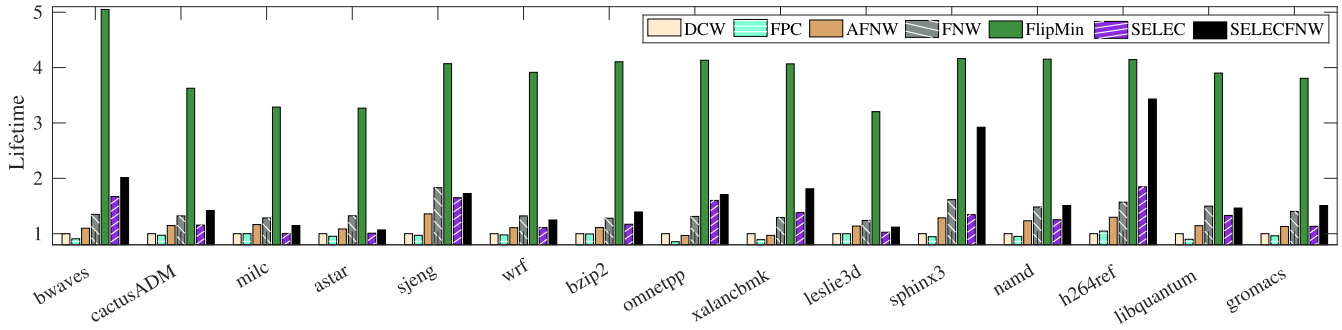


Fig. 19. Comparison of the normalized lifetime.

13.0%, and 22.6%, respectively. In the SELECFNW scheme, the energy reduction is the most in the sphinx3 benchmark, which is about 58.5%. The energy reduction is the least in the benchmark libquantum. The reason is that the libquantum benchmark has a large amount of redundant cache lines. Although the number of write requests is large, the number of bit flips is very small. The write energy is only a small part of the total energy consumption. For most benchmarks, the energy reduction is similar to the bit flip reduction.

3) Lifetime: The lifetime comparison is shown in Fig. 19. The improvements are −4.5%, 14.9%, 40.1%, 292.6%, 31.2%, and 69.9% in FPC, AFNW, Flip-N-Write, FlipMin, SELEC, and SELECFNW, respectively. The lifetime decreases in FPC due to the increase of bit flips. The lifetime improvement of FlipMin is the highest. The improvement comes from two aspects. On the one hand, FlipMin can significantly reduce the bit flips. On the other hand, FlipMin has additional 100% capacity. The bit flips are spread across twice the main memory capacity. Therefore, the average writes which cells endure are reduced.

4) IPC Performance: We compare SELECFNW with the original DCW scheme to evaluate the IPC performance. When there is a read access, the cache line needs to be decoded in SELECFNW. The decoding of SELECFNW consists of the decoding of Flip-N-Write, FlipMin, BDI, or FPC. The decoding latency is about 1 ns, which is 2% of the read latency of PCM-based main memory. The decoding process will work only if the cache line is compressed. The encoding latency is estimated to be 5 ns. We add the encoding latency to the write latency. The write latency of SELECFNW is $155(150+5)$ ns. Fig. 20 shows the comparison of the IPC performance. On average, the IPC performance of SELECFNW degrades 1.0%

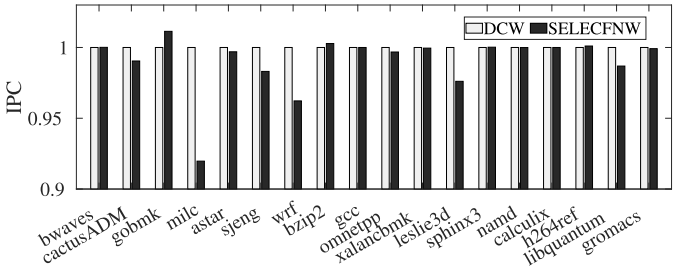


Fig. 20. Comparison of the normalized IPC performance.

due to the encoding latency. For most benchmarks, the IPC performance nearly has no degradation.

VI. Conclusion

NVMs suffer from limited write endurance and high write energy. This article proposes to extend the lifetime of NVMs by combining the data compression techniques with 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 cache line varies and different encoding methods have different tradeoffs between capacity overhead and effect, we select the best-performing data encoding methods according to the size of the space saved by compression. Based on the observation that different compression algorithms can compress different data patterns, we further enhance the compression coverage and compression ratio by dynamically selecting the compression algorithm which leads to the smaller compressed cache line size. To ensure the bit flip reduction of uncompressible cache lines, we enable the encoding of

Flip-N-Write by allowing an additional capacity overhead of about 3.5%. The experimental results show that the proposed scheme can reduce the bit flips by 32.5%, decrease the energy consumption by 22.6% and improve the lifetime by 69.9% with 3.5% capacity overhead.

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 Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1458–1470, Sep. 2017.

[3] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, G. De Micheli, and R. Drechsler, "Endurance management for resistive logic-in-memory computing architectures," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 1092–1097.

[4] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Shenzhen, China, 2013, pp. 222–233.

[5] R. Maddah, S. M. Seyedzadeh, and R. Melhem, "CAFO: Cost aware flip optimization for asymmetric memories," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Burlingame, CA, USA, 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 *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit.*, New York, NY, USA, 2009, pp. 347–357.

[7] S. M. Seyedzadeh, R. Maddah, A. Jones, and R. Melhem, "PRES: Pseudo-random encoding scheme to increase the bit flip reduction in the memory," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 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 *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Dresden, Germany, 2016, pp. 1116–1119.

[9] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Rep. 1500, 2004.

[10] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Tech. (PACT)*, 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 J. Emerg. Technol. Comput. Syst. (JETC)*, vol. 12, no. 4, p. 31, 2016.

[12] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 2–13.

[13] W. S. Zhao *et al.*, "Design and analysis of crossbar architecture based on complementary resistive switching non-volatile memory cells," *J. Parallel Distrib. Comput.*, vol. 74, no. 6, pp. 2484–2496, 2014.

[14] H. Y. Lee *et al.*, "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," in *Proc. Int. Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2010, pp. 1–4.

[15] X. Luo *et al.*, "Enhancing lifetime of NVM-based main memory with bit shifting and flipping," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2014, pp. 1–7.

[16] A. Alsuwaiyan and K. Mohanram, "MFNW: An MLC/TLC flip-N-write architecture," *ACM J. Emerg. Technol. Comput. Syst. (JETC)*, vol. 14, no. 2, p. 28, 2018.

[17] J. Xu, D. Feng, W. Tong, J. Liu, and W. Zhou, "Encoding separately: An energy-efficient write scheme for MLC STT-RAM," in *Proc. Int. Conf. Comput. Design (ICCD)*, Boston, MA, USA, 2017, pp. 581–584.

[18] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. Int. Conf. Comput. Design (ICCD)*, Amherst, MA, USA, 2011, pp. 175–182.

[19] M. Zhao *et al.*, "State asymmetry driven state remapping in phase change memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 1, pp. 27–40, Jan. 2017.

[20] X. Zang, X. Li, L. Dou, Y. Sun, and M. Zhao, "Energy optimization for multi-level cell non-volatile memory using state remapping," *Microprocess. Microsyst.*, vol. 53, pp. 202–212, Aug. 2017.

[21] 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 *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, Paris, France, 2014, pp. 51–56.

[22] P. M. Palangappa and K. Mohanram, "Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories," in *Proc. 25th Ed. Great Lakes Symp. VLSI*, 2015, pp. 221–224.

[23] L. Jiang, Y. Zhang, and J. Yang, "Mitigating write disturbance in super-dense phase change memories," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Atlanta, GA, USA, 2014, pp. 216–227.

[24] A. Jadidi, M. Arjomand, M. K. Tavana, D. R. Kaeli, M. T. Kandemir, and C. R. Das, "Exploring the potential for collaborative data compression and hard-error tolerance in PCM memories," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Denver, CO, USA, 2017, pp. 85–96.

[25] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, and C. Li, "Extending the lifetime of NVMS with compression," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Dresden, Germany, 2018, pp. 1604–1609.

[26] Y. Guo, Y. Hua, and P. Zuo, "A latency-optimized and energy-efficient write scheme in NVM-based main memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 62–74, Jan. 2020.

[27] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, C. Li, and W. Zhou, "Improving performance of TLC RRAM with compression-ratio-aware data encoding," in *Proc. Int. Conf. Comput. Design (ICCD)*, Boston, MA, USA, 2017, pp. 573–580.

[28] P. M. Palangappa and K. Mohanram, "CompEX++: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVMS," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, Apr. 2017, Art. no. 10.

[29] S. Seyedzadeh, A. Jones, and R. Melhem, "Enabling fine-grain restricted coset coding through word-level compression for PCM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 350–361.

[30] S. Asadi, A. M. H. Monazzah, H. Farbeh, and S. G. Miremadi, "WIPE: Wearout informed pattern elimination to improve the endurance of NVM-based caches," in *Proc. 22nd Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2017, pp. 188–193.

[31] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, New Orleans, LA, USA, 2012, pp. 1–10.

[32] M. Arjomand, A. Jadidi, A. Shafiee, and H. Sarbazi-Azad, "A morphable phase change memory architecture considering frequent zero values," in *Proc. Int. Conf. Comput. Design (ICCD)*, Amherst, MA, USA, 2011, pp. 373–380.

[33] L. Jiang, Y. Zhang, and J. Yang, "ER: Elastic RESET for low power and long endurance MLC based phase change memory," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2012, pp. 39–44.

[34] H. G. Lee, S. Baek, J. Kim, and C. Nicopoulos, "A compression-based hybrid MLC/SLC management technique for phase-change memory systems," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (VLSI)*, Amherst, MA, USA, 2012, pp. 386–391.

[35] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[36] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, USA, 2007, pp. 3014–3017.

[37] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[38] M. Poremba, T. Zhang, and Y. Xie, "NVmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul.–Dec. 2015.

[39] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2010, pp. 333–342.

[40] J. Xu *et al.*, "Adaptive granularity encoding for energy-efficient non-volatile main memory," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, p. 114.

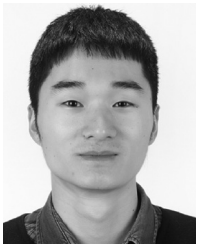
[41] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 141–152. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815980

[42] J. Xu *et al.*, "An efficient spare-line replacement scheme to enhance NVM security," in *Proc. 56th Annu. Design Autom. Conf.*, Las Vegas, NV, USA, 2019, p. 91.
[43] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2009, pp. 14–23.
[44] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 383–394.

Dan Feng (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively.

She is a Professor and the Dean of the School of Computer Science and Technology, HUST. She has more than 100 publications in major journals and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, ACM-TOS, FAST, USENIX ATC, EuroSys, ICDCS, HPDC, SC, ICS, IPDPS, DAC, and DATE. Her research interests include computer architecture, nonvolatile memory technology, distributed and parallel file system, and massive storage system.

Prof. Feng has served as the program committees of multiple international conferences, including SC in 2011 and 2013 and MSST in 2012 and 2015. She is a member of the Association for Computing Machinery and the Chair of Information Storage Technology Committee of Chinese Computer Academy.



Jie Xu received the B.E. degree from the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan, China, in 2014, where he is currently pursuing the Ph.D. degree in computer architecture.

He publishes several papers in major journals and international conferences, including FGCS, ICCD, DATE, DAC, and MSST. His research interest includes embedded development on FPGA and nonvolatile memory.

Yu Hua (Senior Member, IEEE) received the B.E. and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 2001 and 2005, respectively.

He is a Professor with the Huazhong University of Science and Technology, Wuhan. He has more than 100 papers to his credit in major journals and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, OSDI, MICRO, USENIX FAST, USENIX ATC, ACM SoCC, SC, HPDC, ICDCS, IPDPS, and MSST. He serves for multiple international conferences, including ASPLOS, SOSP, USENIX ATC, ICS, RTSS, SoCC, ICDCS, INFOCOM, IPDPS, DAC, MSST, and DATE. His research interests include file systems, cloud storage systems, nonvolatile memory, and big data analytics.

Prof. Hua is the Distinguished Member of CCF, a Senior Member of ACM, and a member of USENIX.

Wei Tong received the B.E., M.E., and Ph.D. degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1999, 2002, and 2011, respectively.

She is an Associate Professor with the Wuhan National Laboratory for Optoelectronics, HUST. She has more than 20 publications in journals and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, ACM TACO, DAC, DATE, ICCD, and ICPP. Her present research interests include computer architecture, nonvolatile memory and storage, and software-defined storage.

Jingning Liu received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1982.

She is a Professor with HUST and engaged in researching and teaching of computer system architecture. She has over 20 publications in journals and international conferences, including ACM TACO, NAS, MSST, and ICA3PP. Her research interests include computer storage network system, high-speed interface and channel technology, embedded system, and FPGA design.

Chunyan Li received the B.E. degree in computer science and technology from the China University of Geosciences, Wuhan, China, in 2016. She is currently pursuing the M.S. degree in computer architecture with the Huazhong University of Science and Technology, Wuhan.

Her research interests include nonvolatile memories and distributed file systems.

Yiran Chen (Fellow, IEEE) received the B.S and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

After five years in industry, he joined the University of Pittsburgh, Pittsburgh, PA, USA, in 2010 as an Assistant Professor and then promoted to an Associate Professor with tenure in 2014, held Bicentennial Alumni Faculty Fellow. He is currently a Tenured Associate Professor with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, and serving as the Director of NSF Industry-University Cooperative Research Center for Alternative Sustainable and Intelligent Computing and the Co-Director of Duke Center for Evolutionary Intelligence, focusing on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems. He has published one book and more than 350 technical publications and has been granted 93 U.S. patents.

Dr. Chen is a recipient of the NSF CAREER Award, the ACM SIGDA Outstanding New Faculty Award, and the Humboldt Research Fellowship for Experienced Researchers. He received 6 best paper awards and 12 best paper nominations from international conferences. He serves or served as an Associate Editor of several IEEE and ACM transactions/journals and served on the technical and organization committees of more than 50 international conferences. He is a Distinguished Member of ACM and a Distinguished Lecturer of IEEE CEDA.