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

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

Abstract—Emerging Non-Volatile 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 encoding method is limited by the capacity overhead of the tag bits. In this paper, we propose to exploit the space saved by compression to store the tag bits of 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 uncompressed cache lines. 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—Encoding, Compression, Bit flips, Lifetime, Non-Volatile Memories.

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 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 work 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 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...
The capacity overhead of Flip-N-Write is only 3.1%. We have low capacity overhead. We give every 32-bit data one tag bit. These uncompressible cache lines, we use Flip-N-Write with a 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:

- 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.
- 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.
- To improve the compression coverage and compression ratio and provide more space for data encoding, we propose to select the compression algorithm which has a smaller compressed cache line size from FPC and BDI.
- To ensure the bit flip reduction of uncompressible cache lines, we propose to combine our technique with Flip-N-Write.
- 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 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 have the advantages of high density, non-volatility and fast read speed. They are considered as promising replacement candidates of traditional DRAM technology. However, they also suffer from high write energy and limited write endurance.

PCMs exploit phase change material such as Ge2Sb2Te2 (GST) to store digital bits. When the GST is 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 $10^6 \sim 10^9$ bit flips [11]. The endurance of PCM is several orders of magnitude 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 ($\text{HfO}_2$, $\text{T}_{2}\text{O}_5$, 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 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 $10^{10}$ 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 Multi-Level Cell (MLC)/Triple-Level Cell (TLC) NVMs by minimizing the cell hamming distance and energy hamming 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. Jue Wang et al. [18] observe that the write energy of MLC PCM is significantly dependent on the cell states. They propose 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. David B. Dgien et al. [21] propose 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. Amin Jadidi et al. [24] exploit 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, our work combines different data encoding methods with multiple data compression algorithms to reduce the bit flips of NVMs. For Triple-Level Cell (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. WLRC [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.

III. MOTIVATION

A. The 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 work. 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 can be 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 fifteen benchmarks selected from SPEC CPU 2006 [35]. In the bwaves benchmark, the sizes of about 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 6 words and 8 words (slightly compressed). For these slightly cache lines, the saved space is limited. Among the fifteen 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.

![Fig. 1. The distribution of the compressed cache line size.](image_url)
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2962127, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Pattern encoded</th>
<th>Example</th>
<th>Compressed</th>
<th>Encoded size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Zero run</td>
<td>0x0000000000000000</td>
<td>0x0</td>
<td>0 bits</td>
</tr>
<tr>
<td>001</td>
<td>8-bits sign-extended</td>
<td>0x000000000000001F</td>
<td>0x7F</td>
<td>8 bits</td>
</tr>
<tr>
<td>010</td>
<td>16-bits sign-extended</td>
<td>0xFFFFFFFFFFFFFFFB6B6</td>
<td>0xB6B6</td>
<td>16 bits</td>
</tr>
<tr>
<td>011</td>
<td>Half-word, padded with a zero half-word</td>
<td>0x000000006543210</td>
<td>0x76543210</td>
<td>32 bits</td>
</tr>
<tr>
<td>100</td>
<td>Two half-words, each two bytes sign-extended</td>
<td>0x7654321000000000</td>
<td>0x76543210</td>
<td>32 bits</td>
</tr>
<tr>
<td>101</td>
<td>Consisting of four repeated double bytes</td>
<td>0xCAFECAFECAFECAFE</td>
<td>0xCAFE</td>
<td>16 bits</td>
</tr>
<tr>
<td>110</td>
<td>Uncompressible</td>
<td>0x0123456789ABCDEF</td>
<td>0x0123456789ABCDEF</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

This table lists the patterns of 64-bit FPC [9], [28].

![Fig. 2. An example of Flip-N-Write.](image)

Fig. 2. An example of Flip-N-Write.

![Fig. 3. An example of FlipMin.](image)

Fig. 3. An example of FlipMin.

In Flip-Min, the vector “1011” reduces 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 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. Figure 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\(\text{ns} \) encoding latency and 0.38\(\text{ns} \) decoding latency [4], while the coset code with 12.5% capacity overhead (RM(1, 7)) incurs 12.86\(\text{ns} \) encoding latency and 0.59\(\text{ns} \) decoding latency [4]. RM(1, 3) and RM(1, 7) belong to Reed-Muller code. Reed-Muller code can be used to generate the coset. The general form of 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.

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.
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 in 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.

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 12.86ns, which is 8.6% of the write latency of PCM. The encoding energy is 63.4pJ [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 work 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 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 Equation (1). In Equation (1), R represents the bit flip reduction, and N equals the integer portion part of D/S.
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 deltas 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 \( \frac{\text{Original Size}}{\text{Compressed Size}} \) 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. Fig. 8 and Fig. 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 bwave, cactusADM, milc, astar, omnetpp, sphinx3, namd and h264ref. The average compression coverages of the fifteen benchmarks are 70.6%, 59.8%, 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.
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 ← FPCCompression(NewLine)
3: LB ← BDICompression(NewLine)
4: LC ← size(LF) < size(LB)? LF : LB
5: /* Selective encoding */
6: D ← size(LC)
7: S ← L = P − D
8: if S == 0 then
9: LE ← LC
10: else if S < D then
11: LE ← Flip − N − WriteEncoding(OldLine, LC);
12: else if S >= D then
13: LE ← FlipMinEncoding(OldLine, LC)
14: end if

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 Fig. 8 and Fig. 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 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 work. 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 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

Fig. 10. The combination of Flip-N-Write with compression encoding.

Fig. 11. The system architecture.
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 11pJ, which is about half of the write energy a PCM cell. The encoding latency is between 2ns and 8ns. The average encoding latency is 5ns, 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Energy</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDI</td>
<td>Encoding 3.9pJ</td>
<td>2ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>FPC</td>
<td>Encoding 2.1pJ</td>
<td>2ns</td>
</tr>
<tr>
<td></td>
<td>Decoding 1.2pJ</td>
<td>1ns</td>
</tr>
<tr>
<td>Selecting Compression</td>
<td>Encoding 0.31pJ</td>
<td>0.43ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>Flip-N-Write</td>
<td>Encoding 6.1pJ</td>
<td>0.54ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>FlipMin</td>
<td>Encoding 8.5pJ</td>
<td>4.09ns</td>
</tr>
<tr>
<td></td>
<td>Decoding 0.3pJ</td>
<td>0.38ns</td>
</tr>
<tr>
<td>Selecting Encoding</td>
<td>Encoding 1.7pJ</td>
<td>1.32ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
</tbody>
</table>

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 2.1pJ/1.2pJ [31]. The latencies of compression and decompression are estimated to be 2ns and 1ns [28]. BDI [10] has low hardware implementation complexity compared with FPC. The compression latency is about 2ns. 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 0.3pJ/8.4pJ [4]. The latency and energy overheads of FlipMin are evaluated with Synopsys Design Compiler and IC Compiler based on the Nangate 45nm semi-custom cell library [4].

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 0.31pJ and 0.43ns. 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 1.52ns, and the energy overhead is 1.7pJ. We get the overhead values by using Synopsys Design Compiler to synthesize the logic in 130nm technology and scaling the results down to 22nm 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.

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 11pJ, which is about half of the write energy a PCM cell. The encoding latency is between 2ns and 8ns. The average encoding latency is 5ns, 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Energy</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDI</td>
<td>Encoding 3.9pJ</td>
<td>2ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>FPC</td>
<td>Encoding 2.1pJ</td>
<td>2ns</td>
</tr>
<tr>
<td></td>
<td>Decoding 1.2pJ</td>
<td>1ns</td>
</tr>
<tr>
<td>Selecting Compression</td>
<td>Encoding 0.31pJ</td>
<td>0.43ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>Flip-N-Write</td>
<td>Encoding 6.1pJ</td>
<td>0.54ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
<tr>
<td>FlipMin</td>
<td>Encoding 8.5pJ</td>
<td>4.09ns</td>
</tr>
<tr>
<td></td>
<td>Decoding 0.3pJ</td>
<td>0.38ns</td>
</tr>
<tr>
<td>Selecting Encoding</td>
<td>Encoding 1.7pJ</td>
<td>1.32ns</td>
</tr>
<tr>
<td></td>
<td>Decoding -</td>
<td>-</td>
</tr>
</tbody>
</table>

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].

<table>
<thead>
<tr>
<th>TABLE IV</th>
<th>SYSTEM CONFIGURATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>4-Core, 2.0GHz, out-of-order</td>
</tr>
<tr>
<td>L1 I/D cache</td>
<td>private, 32KB, 2-way, 2-cycle latency</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>private, 1MB, 8-way, 20-cycle latency</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>shared, 16MB, 16-way, 50-cycle latency</td>
</tr>
<tr>
<td>Memory Controller</td>
<td>FRFCS</td>
</tr>
<tr>
<td>Memory Organization</td>
<td>4GB PCM, Read 50ns, Write 150ns, 20pJ/bit</td>
</tr>
</tbody>
</table>

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.

- **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.  
- **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.  
- **BDI+FNW**: Cache lines are compressed by BDI. Flip-N-Write is used to encode the compressed cache line.  
- **BDI+FlipMin**: Cache lines are compressed by BDI. FlipMin is used to encode the compressed cache line.  
- **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.  
- **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.

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 $0.4\% = \frac{2}{512}$. 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 significantly 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 lines, Flip-N-Write and FlipMin cannot be applied. Therefore, BDI+FNW has the similar bit flips to BDI+FlipMin in 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.

<table>
<thead>
<tr>
<th>TABLE V</th>
<th>THE COMPARISON OF CAPACITY OVERHEAD, BIT FLIPS, ENERGY AND LIFETIME (NORMALIZED TO DCW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity overhead</td>
<td>Bit flips</td>
</tr>
<tr>
<td>FPC+FNW</td>
<td>0.2%</td>
</tr>
<tr>
<td>FPC+FlipMin</td>
<td>0.2%</td>
</tr>
<tr>
<td>BDI+FNW</td>
<td>0.2%</td>
</tr>
<tr>
<td>BDI+FlipMin</td>
<td>0.2%</td>
</tr>
<tr>
<td>COEF</td>
<td>0.2%</td>
</tr>
<tr>
<td>SELEC</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
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 to the number of bit flips, as defined in Equation (2). \( C \) is a constant value for all the schemes.

\[
\text{Lifetime} = \frac{C \times \text{Total Number of Cells}}{\text{Total Number of Bit Flips}} \tag{2}
\]

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.

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.

- DCW [36]: The redundant bit flips are eliminated.
- FPC [9]: The cache line is compressed by FPC at the granularity of 64-bit words.
- AFNW [22]: 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. Each 4-bit data are mapped into a set of 16 8-bit vectors. The vector which causes the minimum bit flips are selected.
• 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.
• 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.

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

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

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%. 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 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 SELC can reduce the similar bit flips to Flip-N-Write. In the aspect of capacity overhead, SELC 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%, 13.0% and 22.6%. 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 1ns, 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 5ns. 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% 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 paper 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 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 uncompressed cache lines, we enable the encoding of Flip-N-Write by allowing an additional capacity overhead of about 3.5%. 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.

ACKNOWLEDGMENT

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, and in part by the Shenzhen Research Funding of Science and Technology under Grant JCYJ20170307172447622. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

REFERENCES


Wei Tong obtained her BE, ME and PhD degrees from Huazhong University of Science and Technology (HUST), China, respectively in 1999, 2002, and 2011. She is an associate professor in Wuhan National Laboratory for Optoelectronics, HUST. Her present research interests include computer architecture, non-volatile memory & storage, and software-defined storage. She has more than 20 publications in journals and international conferences including IEEE TC, ACM TACO, DAC, DATE, ICCD, ICPP.

Dan Feng (M’95) received the B.E., M.E., and Ph.D. degrees in computer science and technology from 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. Her research interests include computer architecture, Non-Volatile memory technology, distributed and parallel file system, and massive storage system. She has more than 100 publications in major journals and international conferences, including IEEE TC, IEEE TPDS, ACM-TOS, FAST, USENIX ATC, EuroSys, ICDCS, HPDC, SC, ICS, IPDPS, DAC and DATE. Dr. Feng has served as the program committee of multiple international conferences, including SC 2011, 2013 and MSST 2012, 2015. She is a member of IEEE and ACM (the Association for Computing Machinery), chair of Information Storage Technology Committee of Chinese Computer academy.

Jingning Liu received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1982. She is a professor in the HUST and engaged in researching and teaching of computer system architecture. Her research interests include computer storage network systems, high-speed interface and channel technology, embedded system and FPGA design. She has over 20 publications in journals and international conferences including ACM TACO, NAS, MSST and ICA3PP.

Jie Xu received the B.E. degree in School of Optical and Electronic Information (OEI) from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently working toward the Ph.D. degree in computer architecture from HUST. His research interest includes embedded development on FPGA and non-volatile memory. He publishes several papers in major journals and international conferences, including FGCS, ICCD, DATE, DAC and MSST.

Chunyan Li received the B.E. degree in computer science and technology from China University of Geosciences (CUG), Wuhan, China, in 2016. She is currently working toward the M.S. degree in computer architecture from HUST. Her research interests include non-volatile memories and distributed file systems.

Yiran Chen (M’04-SM’16-F’18) received B.S and M.S. from Tsinghua University and Ph.D. from Purdue University in 2005. After five years in industry, he joined University of Pittsburgh in 2010 as Assistant Professor and then promoted to Associate Professor with tenure in 2014, held Bicentennial Alumni Faculty Fellow. He now is a tenured Associate Professor of the Department of Electrical and Computer Engineering at Duke University and serving as the director of NSF Industry–University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC) and co-director of Duke Center for Evolutionary Intelligence (CEI), focusing on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems. Dr. Chen has published one book and more than 350 technical publications and has been granted 93 US patents. He serves or served the associate editor of several IEEE and ACM transactions/journals and served on the technical and organization committees of more than 50 international conferences. He received 6 best paper awards and 12 best paper nominations from international conferences. He is the recipient of NSF CAREER award and ACM SIGDA outstanding new faculty award. He is the Fellow of IEEE and Distinguished Member of ACM, a distinguished lecturer of IEEE CEDA, and the recipient of the Humboldt Research Fellowship for Experienced Researchers.