A Latency-optimized and Energy-efficient Write Scheme in NVM-based Main Memory

Yuncheng Guo, Yu Hua, *Senior Member, IEEE,* and Pengfei Zuo, *Student Member, IEEE*

***Abstract*—Non-volatile memory technologies (NVMs) are promising candidates as the next-generation main memory due to high scalability and low energy consumption. However, the performance bottlenecks, such as high write latency and low cell endurance, still exist in NVMs. To address these problems, frequent pattern compression schemes have been widely used, which however suffer from the lack of flexibility and adaptability. In order to overcome these shortcomings, we propose a well-adaptive NVM write scheme, called Dynamic Frequent Pattern Compression (DFPC), to significantly reduce the amount of write units and extend the lifetime. Instead of only using static frequent patterns in existing FPC schemes, which are pre-defined and not always efficient for all applications, the idea behind DFPC is to exploit the characteristics of data distribution in execution to obtain dynamic patterns, which often appear in the real-world applications. To further improve the compression ratio, we exploit the value locality in a cache line to extend the granularity of dynamic patterns. Hence DFPC can encode the contents of cache lines with more kinds of frequent data patterns. Moreover, to further support efficient write and read operations in the context of MLC/TLC NVMs, we need to extend the DFPC to improve performance in terms of the access latency and energy consumption. We hence propose a latency-optimized and energy-efficient compression write scheme to encode the compressed data with low energy and latency states, i.e., Enhanced Dynamic Frequent Pattern Compression (EDFPC), thus reducing the latency and energy consumption. We implement DFPC in GEM5 with NVMain and execute the applications from SPEC CPU2006 to evaluate our scheme. Experimental results demonstrate the efficacy and efficiency of DFPC. We have released the source codes for public use at Github *https://github.com/dfpcscheme/DFPCScheme*.**

***Index Terms*—Non-volatile memory, Compression, Encoder, Frequent pattern.**

I. Introduction

With the rapid growth of massive data to be processed, there is increasing demand to deploy large main memories [1]. However, the traditional DRAM technology as main memory is facing significant challenges in the cell scalability and power leakage. Non-volatile memories (NVMs), such as phase change memory (PCM) and resistive random access memory (ReRAM), have the potential to build future memory systems due to their salient features of lower standby power consumption and better scalability than DRAM [2], [3], [4], [5], [6], [7], [8]. However, NVMs suffer from a number of shortcomings in performance compared to DRAM [9], [10], [11], [12], [13], [14], [15], [16], [17]. In practice, we need to carefully handle the problems of write latency and limited lifetime in NVMs[18], [19]. For example, the write latency of PCM cells is about 150 to 220 ns, much more than the write latency of DRAM with 50 ns. As for lifetime, PCM cells can be only written $10^7 - 10^8$ times, compared with the DRAM cells with 10^{15} writes [18]. In addition, Multi-level/Triple-level cell PCMs (MLC/TLC PCMs) have been put forward to offer higher capacity, which further reduce the cell stability, and thus increase latencies and reduce cell lifetime [12], [20], [21], [22], [23], [24].

In order to improve the write performance and endurance of NVMs, existing schemes mainly consider to reduce the number of write units. For example, Flip-N-Write (FNW) [25] compares the old and new data to reduce the number of bits changed by at least half. The efficiency of FNW depends on the difference between the old and new data in each cache line, which fails to exploit the data redundancy among different cache lines. In order to further reduce the data to be written, Frequent Pattern Compression (sFPC) [26] reduces the number of the written bits via compressing each 32-bit word. Specifically, sFPC maintains a pattern table in the memory controller and the table contains multiple common word patterns, such as the full-zero word pattern. If matching any of the patterns in the table, each word in the cache line is encoded into a compressed format, thus reducing the number of written bits. The word patterns in the pattern table are pre-set and cannot be modified, called *static patterns*. Dgien et al. [27] propose FPC that combines sFPC and FNW, and show that FPC can reduce on average 2× more bit-writes than FNW. However, the FPC cannot work for the applications in which the word patterns with high frequency of appearance are not matched with the static patterns.

In order to improve the flexibility and adaptability of data patterns, we analyze the distribution of data values in each word and the word patterns with high frequency of appearance in different applications. We observe that different applications have their own high-frequent word patterns that may not be included in the pattern table. We call these high-frequent patterns that are different from application to application *dynamic patterns*. Based on the observation, we propose a well-adaptive NVM write scheme, called Dynamic Frequent Pattern Compression (DFPC) [28]. The idea behind DFPC is to extend the pattern table to include the dynamic patterns besides the static patterns. The dynamic patterns are obtained via sampling and analyzing the characteristic of bit value distribution in the cache lines to be written to NVM. We reduce the overhead of sampling by utilizing the match operations in the compression scheme. Moreover, DFPC exploits the

Y. Guo, Y. Hua and P. Zuo are with Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {ycguo, csyhua, pfzuo}@hust.edu.cn. (Corresponding author: Yu Hua.)

value locality in a cache line to extend the granularity of dynamic patterns, from 32 bits to 64 bytes. Specifically, our contributions are summarized as follows.

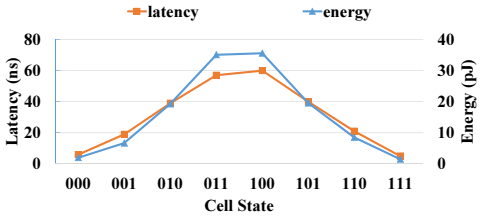
- **Pattern Extraction at Runtime.** In contrast to previous frequent pattern compression schemes, which require extensive statistical analysis to determine the patterns, our proposed extracting algorithm obtains dynamic patterns by sampling and analyzing the incoming data at runtime, and hence improves the flexibility and adaptability of the compression scheme in a cost-efficient manner.
- **An Efficient NVM Write Scheme.** We propose a write-efficient NVM-based scheme, called DFPC. DFPC consists of two important parts, including dynamic patterns and extended patterns. By sampling the distribution of zero-characters and extracting the potential dynamic data patterns, DFPC compresses write data with dynamic patterns to improve the adaptation of the compression. For further reducing the amount of write-bits, DFPC extends the data pattern for compressing the all-zero cache lines and exploiting value locality. DFPC improves the system performance in terms of read and write latency.
- **A Latency-optimized and Energy-efficient DFPC for MLC/TLC NVMs.** We propose a latency-optimized and energy-efficient write scheme for MLC/TLC NVMs, called Enhanced Dynamic Frequent Pattern Compression (EDFPC), to further reduce the latency and energy consumption. EDFPC leverages the different physical properties of the cell states in MLC/TLC NVMs and uses the free space in the cache lines after compressions. EDFPC encodes the compressed data in a cache line with lower energy and latency states of all the cell states to reduce the write latency and energy consumption. Hence, EDFPC can accelerate the processing of instructions and reduce the total energy consumption.
- **Real Implementation in GEM5.** We have implemented DFPC and EDFPC, in GEM5 [29] with NVMain [30] and evaluated it using SPEC CPU2006 benchmark suites. Experiments demonstrate that DFPC has less extra overheads and reduces more write-bits, compared with the state-of-the-art work, Data Comparison Write (DCW), Flip-N-Write (FNW), Base-Delta-Immediate Compression combined with FNW (BDI), and Frequent Pattern Compression combined with FNW (FPC). We have released the source codes at Github.

The rest of this paper is organized as follows. Section II introduces the background and our observations. Section III presents the details of system design in DFPC. Section IV presents the design of EDFPC. Section V shows the performance evaluation. Section VI presents related work. Finally, we conclude this paper in Section VII.

II. Backgrounds and Motivations

A. NVM Properties and Limitations

Unlike traditional charge-based memories such as DRAM and SRAM, emerging non-volatile memories store data by using resistive memories, which have higher density and scalability. Hence, NVMs have the potential to be widely used in the main memory.



Cell State	000	001	010	011	100	101	110	111
Latency (ns)	6	19	39	57	60	40	21	5
Energy (pJ)	2	6.7	19.3	35.1	35.6	19.6	8.5	1.5

Fig. 1. Energy and latency for TLC NVMs P&V [20], [31].

Since all NVMs store information by changing the physical states, the write operation consumes longer time and energy than the read operation, which leads to the asymmetry of read and write. Moreover, the write operation wears out the NVM cell especially in high frequency, which results in the limited endurance of NVMs. Therefore, NVM-based systems need to reduce the number of write-bits in the write operation. As one of the promising non-volatile memory technologies, PCM technology uses the properties in resistance of chalcogenide glass to store data [9]. The phase change material used is the alloy of germanium, antimony, and tellurium, such as $Ge_2Sb_2Te_5$ (GST). The material has two states, crystalline (SET) state and amorphous (RESET) state. The resistances of the material with different states are drastically different. In general, the resistance value of the amorphous-state material is much higher than that of the crystalline-state material. PCM stores binary informantion with two states by using the resistance gap of the material. To perform a RESET (SET) operation to write '0' ('1') to the PCM cell, a PCM cell is heated above the melting point to melt the chalcogenide material (above its crystallization temperature but below its melting point) followed by fast (slow) cooling to change the state. The ReRAM cell uses the insulating dielectric within the Metal-Insulator-Metal structure. By using suitable voltages, it may be changed between low-resistance state (SET) and high-resistance state (RESET).

B. Energy and Latency of MLC/TLC NVMs

Both PCM and RRAM support multi-level/triple-level cell (MLC/TLC) technology, due to the large gap between the lowest-resistance state (SET) and highest-resistance state (RESET). For example, a cell of TLC NVMs can be programed to 8 stable resistance levels. Hence, TLC NVMs can store 3 logical bits of data in each cell. With this programming scheme, MLC/TLC NVMs can store more data in a fixed-size device and increase the data density and reach higher capacity [32]. However, MLC/TLC cells have various states with different physical properties. Compared with SLC NVMs, the programming for MLC/TLC NVMs is more complicated. There are two state-of-the-art program-and-verify (P&V) operations, reset-to-set (RTS) and set-to-reset (STR) [33], [34]. We illustrate P&V for TLC NVM module in Figure 1. In RTS, one reset operation is first applied to make the cell fully amorphous

Fig. 2. An illustration of different 16-bit write operations. The data bits in red color need to be written. The blue data bits signify the matching prefix.

to the reset state, “000”, and then a series of set operations with various iterations are applied to reach the target state. Similarly, in STR, one set operation is first applied to make the cell to the set state, “111”, and then various iterations of reset operations are applied to reach the target state. In general, RTS is used to program states “000” to “011” and STR is used to program states “111” to “100”. As shown in Figure 1, the required energy and latency for programming a cell of TLC NVMs to states “011” and “100” are about 10 to 20 times in comparison to the reset “000” and set states “111” [20], [31].

C. DCW, FNW and FPC

Data Comparison Write (DCW) [35] exploits the non-volatility of NVMs and checks the new write-bits with existing bits to determine the modified bits. Only the modified bits require a write operation. Flip-N-Write (FNW) [25] compares the new write-bits with existing bits to reduce the number of bit-flips. FNW reduces the number of bits changed more than half by using only one tag bit per compared unit.

Compression is an attractive approach to increase effective memory capacity of a large-size memory. Compression increases space usage, and hence improves performance and energy efficiency. Alameldeen et al. propose Frequent Pattern Compression (sFPC) [26] to compress write words based on a set of static patterns recorded in a pattern table.

Dgien et al. propose FPC by combining sFPC with FNW [27]. This method uses two tag bits per word and reduces the size of data before being written into the data array. Figure 2 presents an example of a write operation with different write schemes. We take 16-bit write unit as an example. In DCW and FNW, the new data will be compared with the data in memory for obtaining the different data bits. In this example, the number of different bits is 6, which is smaller than the half of the total bits of the write unit. So 6 bits will be written by using DCW and FNW. As for FPC, we use a simple pattern table to give an example. The pattern table uses 2-bit prefix with 4 patterns, and each pattern contains 4 bits to represent the 4-bit characters’ status (compressible or incompressible). In Figure 2, since the 2nd and 4th characters of the new data are zero-characters, the Compression Engine matches the new data with the pattern “X0X0" and encodes the prefix of this pattern “01" with the incompressible characters. Then the compressed data with 10 bits will be written in data array based on FNW. Finally, this write operation only writes 4 bits with FPC (sFPC combined with FNW).

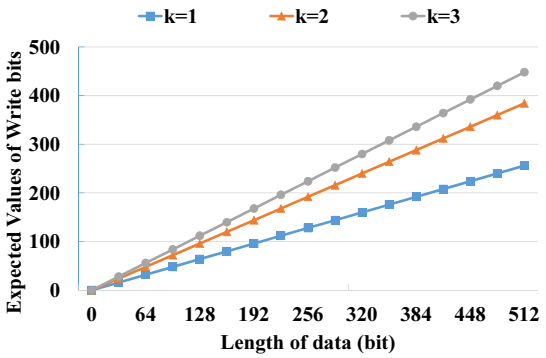


Fig. 3. The expected value of write-bits.

D. Extra Writing in MLC/TLC NVMs

Nowadays, Multi-level/Triple-level cell NVMs (MLC/TLC NVMs) have been put forward to offer higher capacity, which further increase the latencies and reduce cell lifetime [21], [31]. DFPC will have the same advantages in MLC/TLC NVMs. However, compared with SLC cells, TLC cells have more bits and states. Unlike FNW, DCW is the first choice for writing. If there exist only one or two pairs of different bits in a TLC cell between old and new data, the whole cell with 3 bits will be written, which we call Extra Writing in this paper. Extra writing increases the probability of failure in DCW and the number of written bits. To demonstrate our view, as shown in Equation 1,

$$E(N) = k \times ((1 - \frac{1}{2^k}) \left\lfloor \frac{l}{k} \right\rfloor + (1 - \frac{1}{2^{l\%k}})) \quad (1)$$

we calculate the expected values of the number of written bits after DCW in SLC and MLC NVMs, and present them in Figure 3. We consider all of the data bits in new data and existing data in memory are random. k is the number of bits in each cell, and l is the length of data. Two k-bit cells have a $1/k$ chance to be the same. At this moment DCW will take effect. When the number of bits in a cell increases, DCW will have low performance. Moreover, the compression schemes have impact on DCW, which changes the positions of some units, mitigates the efficiency of existing data structures and increases the probabilities of extra writing after DCW. Figure 4 presents an example of the comparison between SLC and TLC NVMs. In SLC NVMs, after compression and DCW, the number of written bits decreases from 8 (only with DCW) to 4. However, in TLC NVMs, the number of written bits increases from 10 (3 cells and 1 bit) to 12 (4 cells). In these cases, compressions deliver poor performance via DCW. We will show the detailed evaluations in Section V-B6.

E. Motivations

Existing schemes [26], [27] generally exploit static patterns for compression, which are obtained by analyzing the statistical characteristics of real-world applications. In general, the frequency of selected patterns directly affects the number of compressible cache lines. However, the high-frequent patterns in the applications are not always matched with the static

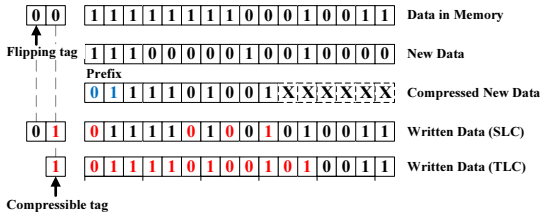


Fig. 4. An illustration of the comparison of the 16-bit write operations in SLC and TLC NVMs.

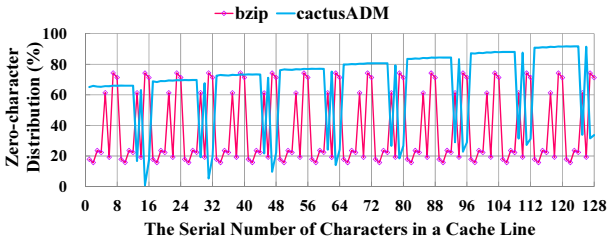


Fig. 5. The distribution of zero-characters.

patterns. In order to improve the patterns used in compression schemes to adapt to different applications, we study the characteristics of the data generated by applications. We observe that most integers used in applications are compressed and stored in a 32-bit form, in which only 4, 8, or 16 bits have been really used. The low utilization results in the wide distribution of zero (in 4-bit character) [36]. Since read and write operations access one cache line (64 Bytes) at a time, the zero-character distribution of the cache line can fully exhibit the data distribution of applications. To verify this point, we conduct the simulation of examining write accesses. We use the applications from SPEC CPU2006 for mimicking the real-world workloads. The results reveal that the potential patterns of the zero-character distribution. We observe that the characters in some specific serial numbers, where the zero-character appears frequently, form a part of a periodic cycle. As shown in Figure 5, each 8-character data corresponds to a periodic cycle in *bzip*. In each periodic cycle, the 4th, 7th, and 8th characters have high frequency of '0'. In *cactusADM*, the specific serial numbers are different with those in *bzip* since the data patterns have changed. If we consider zero-character as the compressible unit, the higher points which represent the higher frequent occurrence of zero-character will be regarded as the compressible part of data pattern. The data patterns of $\frac{64\times8bit}{32bit} = 16$ words in the cache line can be extracted based on the distribution at runtime.

III. System Design and Implementation

In order to describe the design of DFPC, we first define the dynamic pattern extracted from the applications and explain its high compressibility and the feasibility of pattern extraction. We then introduce the extended pattern for improving the efficiency of write operations. Finally, we present a well-adaptive NVM write scheme, called DFPC, and the system architecture as well as the workflow of DFPC.

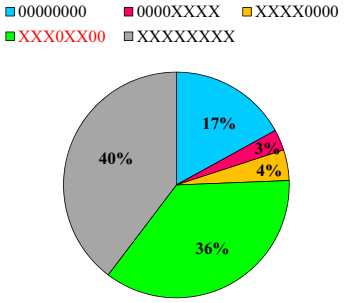


Fig. 6. The distribution of data patterns in *bzip*. The first three patterns are static patterns. "XXX0XX00" is a dynamic pattern extracted from the distribution. The pattern with all 'X' means the incompressible word.

A. Dynamic Pattern

Based on our observation, we first define the static pattern and the dynamic pattern. The static pattern is interpreted as the pattern pre-defined that fails to be modified at runtime. The dynamic pattern is interpreted as the pattern obtained by sampling and analyzing the characteristic of bit value distribution in the incoming data at runtime. Figure 6 presents an example showing the pattern extracted from the distribution in *bzip*. In this example, the dynamic pattern, "XXX0XX00" (the incompressible 4-bit character is expressed as 'X'), can be obviously obtained by observing the zero-character distribution. This pattern occupies 36% of the words, more than other static patterns, which reveals the potential high frequency of the dynamic patterns. We can add the dynamic patterns to the pattern table to enhance the compressibility of the compression scheme.

B. Extended Pattern

Dynamic patterns can improve the adaptation of compression in various applications. However, the scalability of data patterns in compression is low due to the fixed format of data patterns. In our observation, the requests always access one 64-Byte cache line at a time, but FPC only compresses words with 32-bit form. When a write request with all zero bits accesses the data array, it will be divided into 16 words for compression, which causes $3 \times 16 = 48$ write bits before being written. For further reducing the number of bit-writes with high efficiency, we extend the size of dynamic patterns. We improve the traditional extract algorithm to detect more complex patterns, such as BDI patterns, which exploit the value locality of the cache line [37], [38].

C. Dynamic Frequent Pattern Compression

By using the extended dynamic patterns, we propose a well-adaptive NVM write scheme. The idea is to extract the dynamic patterns from the zero-character distribution and use both static patterns and dynamic patterns to compress more data. The scheme is implemented in the Memory Controller. For the compression with common patterns, the scheme divides the 64-Byte cache line into 16 32-bit words. The content of a word is checked to determine if matching any of the patterns with high frequency. Each data pattern contains 8 bits

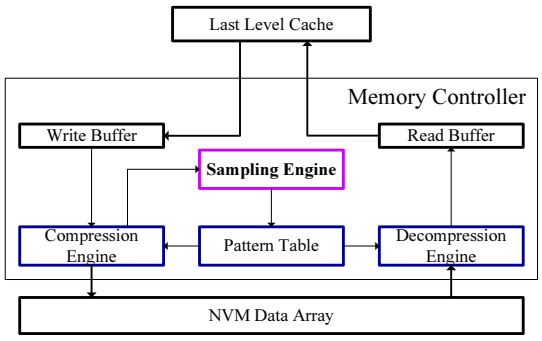


Fig. 7. The DFPC system.

to represent 8 4-bit characters' status (compressible or incompressible). For the compression with the extended patterns, the scheme can select appropriate matching algorithms. For supporting compression in NVMs, the cache line is modified to include the tag bits. Each word needs 2 tag bits as the compressible tag and flipping tag, and hence 32 additional tag bits are added into a 64-Byte cache line. Figure 7 shows the architecture of the NVM-based main memory system with DFPC. We add a component in the architecture of compression scheme, called Sampling Engine, which consists of a group of counters for sampling the zero-character distribution of the application and extracting the dynamic patterns. After obtaining the dynamic patterns, the Sampling Engine adds the patterns to the dynamic pattern table. For write accesses, the scheme consists of three stages, i.e., Sampling stage, Analysis stage and Dual-Pattern Compression stage.

1) Sampling Stage: The dynamic patterns require sufficient number of words to be obtained by the extraction algorithm. To facilitate compression operations in the Sampling stage, a few static patterns are pre-defined in a pattern table. In our experiments, we use 4 static patterns and 3-bit prefix to maintain 8 data patterns in total. So the number of appropriate dynamic patterns after filtering should be no more than 4. During a write access, if a word is compressible and matched with a pattern, it is encoded along with a prefix. The compressible tag is set to be enabled. Before being written through the cache line of NVMs, the bits of each word are compared with existing bits. If more than half of the bits are different, the whole word will flip, and the flipping tag is set to be enabled. For sampling, we design a Sampling Engine to record the most frequent patterns. The Sampling Engine uses two groups of counters. One group is for counting the number of zero-characters in the cache line in Sampling Stage. The number of counters is set as 128(64 Byte / 4 bits per counter). The other group is for counting the reduced bits of the extended patterns in Analysis Stage(All-zero cache lines are not included). To reduce additional overhead of sampling, we leverage a match operation, which reads the value of each character, and supports the sampling operation in the meantime. Hence the sampling information in the Sampling Engine comes from the Compression Engine rather than the Write Buffer.

2) Analysis Stage: When the sampling amount reaches the sampling granularity N, the dynamic patterns are obtained by analyzing the zero-character distribution. The distribution may be quite complex in some application. So we design a simple and effective pattern extracting algorithm with high efficiency. In the Analysis stage, the counters are analyzed by the extracting algorithm (as shown in Algorithm 1). First, the counters are checked for obtaining the Upper Bound (UB) and the Lower Bound (LB). The value of the threshold T is calculated by using the equation in line 5 of Algorithm 1.

Algorithm 1 Algorithm for extracting dynamic patterns.

Require:
The array of count values in counters, C;
The number of counters, n;
The threshold factor, TF;
Ensure:
The array of dynamic patterns, P;
1: **for** each $i \in [1, n]$ **do**
2: $UB = MAX(UB, C[i])$;
3: $LB = MIN(LB, C[i])$;
4: **end for**
5: $T = LB + (UB - LB) \times TF$
6: **for** each $i \in [1, n]$ **do**
7: **if** $C[i] \geq T$ **then**
8: $C[i] = 0$;
9: **else**
10: $C[i] = 1$;
11: **end if**
12: **end for**
13: extracting $\frac{n}{8}$ dynamic patterns from C and adding them to P
14: filtering: eliminating existed and repeated patterns
15: **return** P.

The threshold factor (TF) and the sampling granularity N are pre-defined via experimental verification in Section V-A.

The dynamic patterns can be extracted by comparing the count values of the counters with the threshold. When the count value is larger than or equal to the threshold, the pattern code is set as '0', which means this character is compressible. Otherwise, the pattern code is set as '1' ('X'). After checking all counters, the algorithm extracts 16 patterns. However, some of the extracted patterns are repeated or useless, like "XXXXXXXX". The repeated patterns and incompressible pattern should be removed. DFPC also filters the extracted patterns in the Analysis stage. The data to be written are encoded by the compression with a prefix based on the data pattern which contains 8 4-bit characters. For a 3-bit prefix, the data pattern with one compressible character '0' can only save $4 - 3 = 1$ bit, which has low efficiency and long latency especially when the compressed data is decoded for reading. In the applications with low distribution of all-zero units, most extracted dynamic patterns have 3 or less '0', which have lower compressibility than the static patterns. In order to make a trade-off between the compressibility and extra overheads, DFPC filters out the extracted patterns with low compressibility (only one '0') in the Analysis stage. In order to make full use of the limited number of prefixes, DFPC selects top-4 patterns based on the compression ratios of these ex-

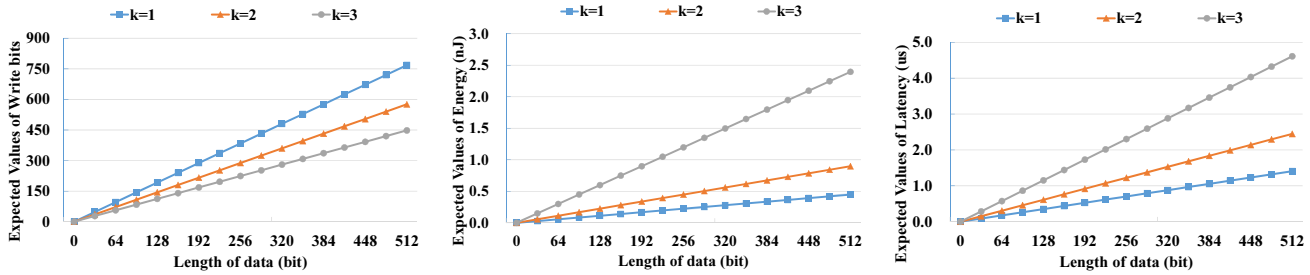


Fig. 8. The expected value of written bits, energy and latency in different kinds of encoding schemes.

tracted patterns and extended patterns. The compression ratio is determined by the pattern's compressibility and frequency of occurrence. Finally, the remaining 4 dynamic patterns are added to the pattern table and won't be modified.

3) Dual-Pattern Compression Stage: In the Dual-Pattern Compression stage, the Compression Engine can obtain two kinds of data patterns from the dynamic pattern table and the static pattern table, respectively. The content of a word is compressed based on the static and dynamic patterns concurrently. The Compression Engine then compares the numbers of compressible bits in two data patterns, and chooses the pattern with more compressible bits as the matching result to execute the compression operation.

During the read accesses, the state of a compressible tag bit determines if the data are compressed. The state of flipping tag bit presents the flipped data. After parsing these tags, the Decompression Engine can decoded the word rapidly. If the data are flipped, the Decompression Engine will first flip the data. Then if the data is compressed, the Decompression Engine will read the first 3 bits to obtain the prefix, which establishes a match between the compressed word and the matching pattern, and find the matching pattern. Finally, the Decompression Engine decodes the compressed data and is filled with zero-characters based on the pattern.

D. Temporal and Spatial Overheads

The implementation of DFPC consists of compression and pattern extraction. The time overhead of compression is about several cycles on average in the Memory Controller [38]. In Section V-A, we define the access time as 3 cycles in compression and 2 cycles in decompression. For pattern extraction in the Analysis Stage, Algorithm 1 traverses the counters with constant-scale time complexity. Since most applications usually run quite a period of time after warming up, the time consumption of pattern extraction can be negligible. After pattern extraction, the static patterns are still retained in the pattern table. Thus the old data compressed by using static patterns can be decompressed via its prefix during the read accesses.

As we mentioned, the cache line is modified to provide the tag bits. FNW needs one flipping tag bit per word. For supporting compressions, like FPC and DFPC, each word needs another tag bit to mark the word compressible or not. In order to carry out the sampling, DFPC needs a set of counters, which only use $128(counters) \times 8\ Byte(64 - bit\ int) = 1KB$ and incur no extra space overhead at run time.

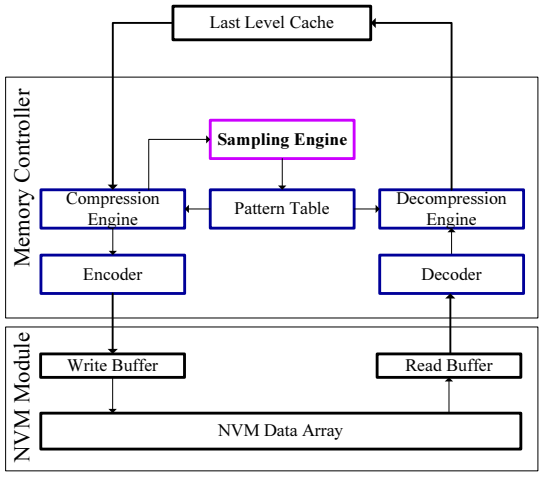


Fig. 9. The EDFPC system.

IV. Enhanced Dynamic Frequent Pattern Compression

A. Latency-optimized and Energy-efficient Encoding Scheme in MLC/TLC Cells

Taking TLC cell as an example, each TLC cell has 8 different states which can store 3 bits of data. As shown in Figure 1, there are huge gaps of energy and latency among the states. We can use several of them for storing data to reduce the energy consumption and latency. However, this encoding scheme will increase the size of data. Fortunately, compression can delete useless information and reduce the size of data, which offers the platform for encoding scheme.

As shown in Figure 8, we calculate the expected values of the number of written bits, energy consumption and write latency in different kinds of encoding schemes (k is the number of bits in each encoded cell). With encoding scheme, we can further reduce the energy consumption and write latency.

B. Enhanced Dynamic Frequent Pattern Compression

By using the encoding scheme after compression, we propose an enhanced DFPC for MLC/TLC NVMs, called EDFPC. We leverage the free space in the cache lines after compressions and encode the compressed data with low energy and latency states. For supporting the encoding scheme, we add a pair of components including an encoder and a decoder

Fig. 10. An illustration of a 16-bit write operation in EDFPC.

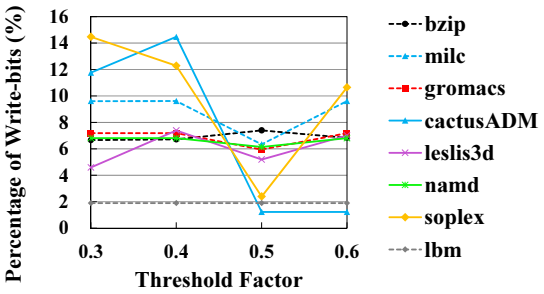


Fig. 11. Write-bits with different Threshold Factors TF.

in Memory Controller. Figure 9 shows the architecture of the MLC/TLC NVM-based main memory system with EDFPC.

After compressing the raw data, if the data from the Compression Engine is incompressible (determined by checking the compressible tag bit) or with low compression ratio, the data will be written directly to NVMs. Otherwise, the Encoder will encode the compressed data to reduce the energy and write latency.

Figure 10 shows an example of the proposed encoding scheme. After compression, the compressed data only has 10 bits but 12 different bits (4 cells) compared with the old data in TLC NVMs, which occurs in Extra Writing. To mitigate the impact of the Extra Writing, as shown in Section IV-A, we select 4 states of the TLC cell with low energy and latency ("000", "001", "110", "111") for encoding. Thus each 2-bit compressed data ("00", "01", "10", "11") will be encoded into 3-bit encoded cell ("000", "001", "110", "111"). In this example, compared with the written data of DFPC with 178 ns write latency and 98.8 pJ energy, the encoded data only costs 90 ns write latency and 33.4 pJ energy, without using DCW. After encoding, the Encoder adds one bit '1' as the encoding suffix for decoding. However, the suffix could create an extra word in the write phase. In the implementation of EDFPC, we merge the 1-bit suffix and the 3-bit compression prefix of DFPC into a 4-bit character to maintain the write order and facilitate the decode operation.

V. Performance Evaluation

In this section, we take PCM as an example of NVMs to demonstrate the performance of DFPC. We evaluate the efficacy and efficiency of our design using the applications from SPEC CPU2006. We first analyze the impact of the parameters, such as Threshold Factor and Sample Granularity. We then present the results of write-bit reduction, write latency, read latency, Instructions Per Cycle (IPC), as well as energy consumption. In TLC NVMs, we point out the poor performance of the compressions and show the improvement of EDFPC.

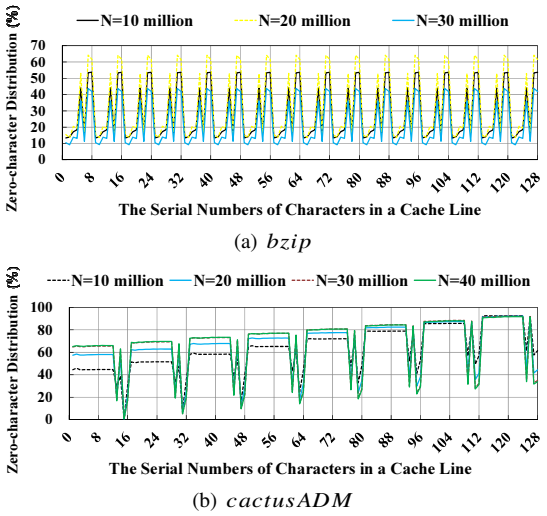
A. Experimental Configurations

We present the configuration and the experimental environment. The GEM5 simulator is a modular platform for computer-system architecture research, and supports a standard PC platform. NVMain is a cycle-accurate main memory simulator designed to simulate emerging non-volatile memories at the architectural level. We implemented DFPC and EDFPC in the GEM5 simulator [29] with NVMain [30] to evaluate our design. The SPEC CPU2006 applications reflect a variety of real integer and floating-point based workloads used by modern computing systems. The configurations of the simulation are shown in Table I. In our experiments, the memory traces from the applications on a machine running two 4-core Intel Xeon E5620 CPUs with 2.4GHz frequency. We also simulate the whole memory hierarchy and adopt two-level cache (L1 and L2 caches). All caches have 64B cache line size. The 8GB PCM main memory has 1 rank and 8 banks and the main memory controller has individual read and write queues and uses FRFCFS scheduling algorithm of NVMain which schedules read requests and only deals with write requests, when the queue of write requests is full. The static patterns ("00000000", "00001111" and "11110000") are selected from the zero-extended patterns in [27], [39]. The read latency of the PCM is 75 ns and the set (reset) latency is 37.5 (12.5) ns per 32-bit word, like the configurations in [39], [40], [41]. During trace generation, the applications are first run through 10 million instructions to warm up the system [38]. The applications then run 2 billion instructions to record enough accesses. We evaluate and compare the performance of our DFPC to three state-of-the-art methods, Flip-N-Write (FNW) [25], Base-Delta-Immediate Compression with FNW (BDI), and Frequent Pattern Compression with FNW (FPC) [27]. In TLC PCMs, we replace FNW with DCW. The latency and energy of each state in the TLC cell follows the parameters in [31], [20]

1) Threshold Factor: Before performing the evaluation of DFPC, we first analyze the impact of the parameters in the design. The threshold factor TF is an important parameter of

TABLE I
Experiment Configurations

Processor and Cache	
CPU	4 cores x86-64 processor, 2 GHz
Private L1/L2 caches	32 KB/ 2048 KB
Memory Using PCM-based Memory	
Capacity	8 GB, 1 channel, 1 ranks, 8 banks
Read latency	75 ns
Set latency	37.5 ns for each word of 32 bits
Reset latency	12.5 ns for each word of 32 bits
Parameters of DFPC	
Compression latency	1.5 ns [38]
Decompression latency	1 ns [38]
Sample granularity N	5 million write accesses
Threshold factor TF	0.5



(a) *bzip*

(b) *cactusADM*

Fig. 12. Zero-character distribution with different Sample Granularities.

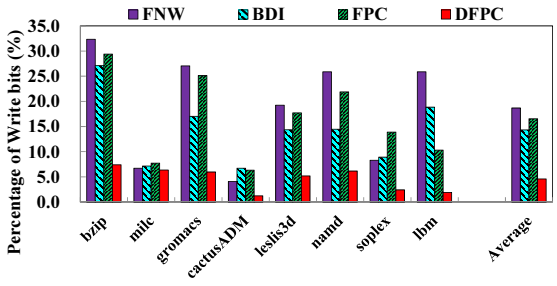


Fig. 13. The percentage of write-bits in SLC NVMs.

the extraction algorithm used in DFPC. With a high threshold factor, the dynamic patterns extracted from the distribution can compress more words due to the less number of specific serial numbers. The dynamic patterns extracted by pre-setting lower threshold factor have more compressible characters in a word. We consider that if we preset the threshold factor within a rational range, there will be a small effect on the performance of write-bits reduction. To verify this point, we implement DFPC and count the amount of written bits in 8 applications with the threshold factor from 0.3 to 0.6. Statistical results are shown in Figure 11. We observe that the reductions of write-bits perform well from 0.4 to 0.6. Hence, we choose 0.5 as the default threshold factor.

2) Sample Granularity: To improve the representation of the dynamic patterns, we need to select appropriate sample granularity N. We use DFPC to collect the zero-character distribution every 10 million write accesses. We present the change of zero-character distribution with the sample granularity N in Figure 12, and the fluctuant number of compressible characters in the cache line within a narrow range is correlated with the result of the extraction algorithm. We observe that there are few differences among the statistics. The cycle of distribution remains unchanged in all statistical results, while the number of compressible characters in a word fluctuates within a narrow range. The dynamic patterns will be more representative when expanding the sample granularity, while the dynamic patterns will be extracted fairly later and perhaps cause performance degradation.

B. The Comparisons between DFPC and Existing NVM Write Schemes

1) Write-bit Reduction: DFPC compresses the contents of words, based on the static and dynamic patterns, to reduce the number of the bits to be written. The write-bit reduction reduces the energy consumption and improves the endurance of PCM. As shown in Figure 13, by using FNW, FPC and BDI can reduce more than 70% written bits. With dynamic patterns, DFPC outperforms FNW, FPC and BDI, and decreases 75%, 72% and 68% written bits on average.

2) Write Latency: Compression schemes can significantly reduce the number of written bits before executing comparison in a write operation. Hence the number of writing words can also be reduced before being written. The write accesses in compressions can be completed more quickly compared with FNW. Experimental results of write latency are shown in Figure 14. DFPC encodes the all-zero cache line from 64 bits to 3 bits, which can be written as one word, and hence improves the performance of write accesses. All of the applications have lower write latency compared with FNW, which demonstrates that FNW indeed has impact on the write-bit reduction of compression schemes. In general, DFPC outperforms FNW, FPC and BDI, and decreases 74%, 43% and 61% write latency on average.

3) Read Latency: Read latency is important to the PCM-based main memory performance. Compression schemes reduce the write latency, and hence improve the efficiency of the queue of operations and reduce the waiting time of read operations, which is an important fraction of read latency. Figure 15 shows the read latency reductions of DFPC, BDI, and FPC compared with FNW. By using compressions, one read phase can gain more data with the decreasing size of the compressed word, and the experimental results verify that the performance of write-bits reduction impacts on read latency. Due to the improvement in write latency, the experimental results of DFPC in read latency outperform BDI and FPC. DFPC obtains 38% more read latency reduction compared with FNW.

4) IPC Improvement: IPC is important to the entire system performance. Minimizing the number of write-bits can reduce the response time and improve the access speed. IPC is influenced by a variety of factors, including the latency of the accesses. The results of IPC improvement are illustrated in Figure 16. DFPC performs well in most applications, especially in *bzip* and *lbm* due to the high compressibility of dynamic patterns and high frequency of write accesses. DFPC obtains 23% speedup on average, compared with FNW.

5) Energy Improvement: Energy consumption is an important concern in the PCM-based main memory, especially in MLC and TLC PCMs. In write accesses, the PCM cell requires high current. Handling the high levels of power is a crucial challenge for the PCM devices and the entire system, especially during the high frequency of write accesses. High energy consumption causes the heat problem, which decreases

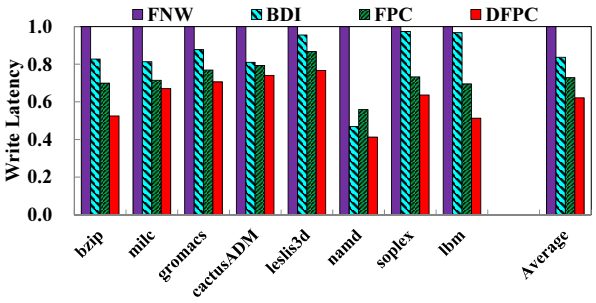


Fig. 14. Write latency reduction in SLC NVMs (normalized to FNW).

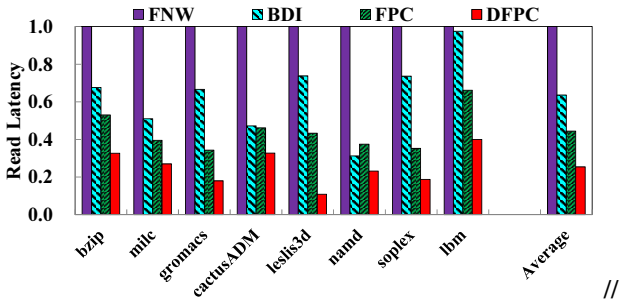


Fig. 15. Read latency reduction in SLC NVMs (normalized to FNW).

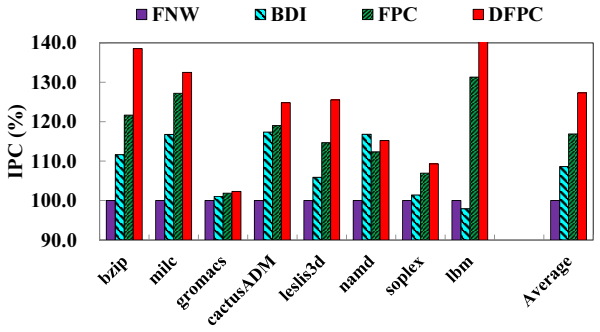


Fig. 16. IPC improvement in SLC NVMs (normalized to FNW).

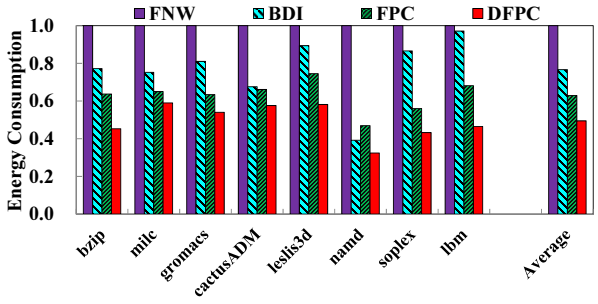


Fig. 17. Energy consumption reduction in SLC NVMs (normalized to FNW).

the endurance and possibly destroys the device, and increases the budget for cooling. Hence the improvement on energy can bring significant benefits to both the budget saving and environment. As shown in Figure 17, DFPC alleviates the energy consumption and extends the lifetime. On average, DFPC saves 41% energy compared with FNW.

6) The Performance Metrics in TLC NVMs: We compare DFPC and existing compression schemes in TLC NVMs. Unlike SLC NVMs, it is inefficient to deploy FNW in the TLC NVM modules since the number of cell states is more than 2. In the evaluation, we use DCW before writing new data to the data array. Figures 18, 19, 20, 22 and 21 present the evaluation results. We observe that compressions do not work well as expected. In most of the applications, FPC has the lowest performance among the three compressions, even poorer than baseline (DCW). As for BDI and DFPC, the two compression schemes do reduce the number of the written bits, but there are not dramatic improvements in other metrics. We count the compression ratios of the three compression schemes based on the Equation 2 and analysis the reason of the performance reduction.

$$Compression\ Ratio = \frac{Original\ Size}{Compression\ Size} \tag{2}$$

As shown in Figure 23, DFPC outperforms the other two compression schemes in the write-bit reduction before writing. However, DFPC delivers poor performance with DCW due to the extra writes. The word-level compressions, such as FPC and part of DFPC, divide a cache line into 16 32-bit words and match each word with a data pattern in the pattern table, which dramatically changes the structures of new data. In addition, BDI does not provide the high compression ratio, but with compressing the whole cache line rather than 32-bit word, BDI achieves the highest write performance of the three compression schemes. These evaluation results verify our argument in Section II-D that compression increases the probabilities of extra writing in TLC NVMs and leads to performance degradation after DCW.

C. The Comparisons between DFPC and EDFPC in TLC NVMs

In this subsection, we compare DFPC and EDFPC in terms of write-bit reduction, latency, IPC and energy consumption. We deploy the simulator in CPU-memory mode and use 10 applications of SPEC CPU2006 and set the parameters of NVM based on [20], [31]. We use DCW before writing data to the NVM data array to reduce the energy consumption and extend the lifetime of NVM devices.

1) Written Bits: In Section IV-A, we have explained the potential write-bit increase in theory when the data bits in the cells are all random. To show the acceptable overhead of written bits, we evaluate the changes of the written bits. As shown in Figure 18, in most of the applications, EDFPC performs well without many extra data bits, which seems to be contrary to the expected results. After compression, the cell bits of the compressed data mitigate most of the redundancies and tend to be random, which meets the requirements in the equation of the theoretical model in Section IV-A. However, as part of the space overhead in compressions, the 3-bit prefix can only represent 8 data patterns. There are still redundant data bits in the compressed data. When writing the compressed data with TLC mode (3 bits per cell), none of the write units, such as 4-bit character, 8-bit byte and 32-bit word,

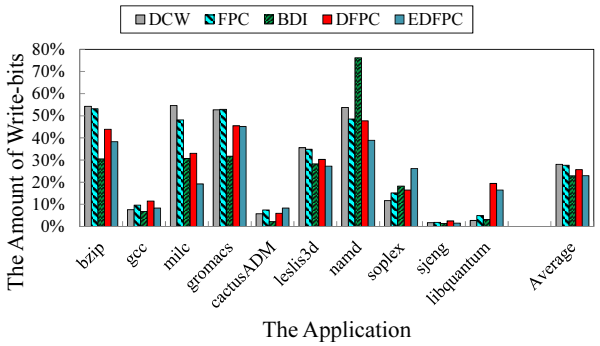


Fig. 18. The percentage of write-bits in TLC NVMs.

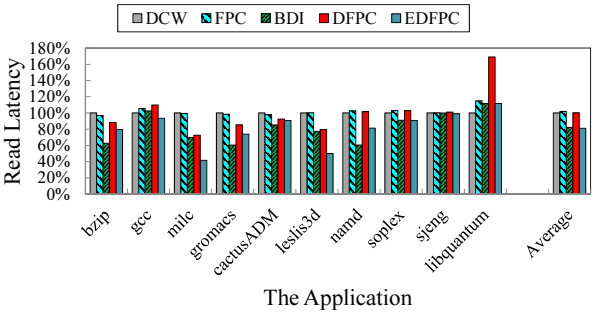


Fig. 19. Read latency in TLC NVMs (normalized to DCW).

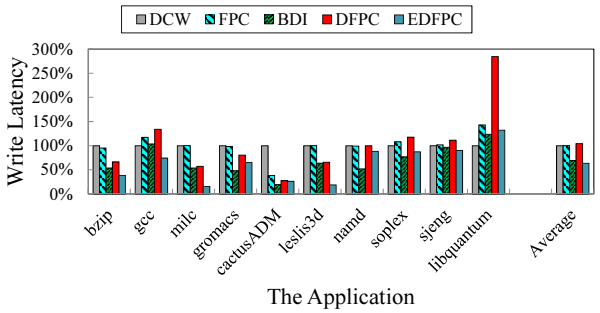


Fig. 20. Write latency in TLC NVMs (normalized to DCW).

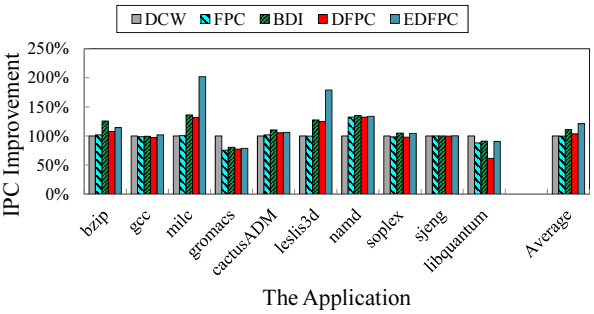


Fig. 21. IPC in TLC NVMs (normalized to DCW).

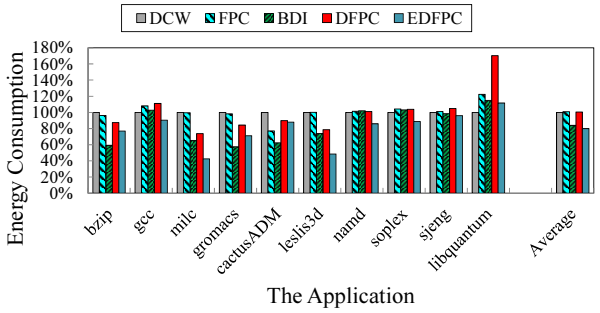


Fig. 22. Energy consumption in TLC NVMs (normalized to DCW).

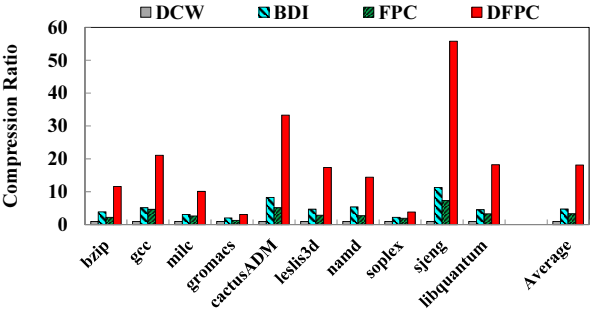


Fig. 23. Compression ratio comparisons.

can be just divided without being combined with other ones, which disrupts the correlation within the write units after compression, increases the probabilities of different cell data and occurs extra writing. On the contrary, when writing the encoded compressed data with SLC (MLC) mode (1 (2) bit(s) per cell), all of the 4-bit character, 8-bit byte and 32-bit word can be just divided into 4 (2), 8 (4), 32 (16) cells respectively. Each cell still reserves the redundant data which is beneficial for DCW but being omitted by the compressions. With an encoder, EDFPC decreases 10%, 18% written bits on average, compared with DFPC and DCW.

2) Write Latency: To show the benefits of EDFPC, we use DCW without any compression schemes as the baseline and normalize the experimental results. The average write latency is shown in Figure 20. We can observe that EDFPC reduces the write latency by about 39% compared with DFPC. There are two reasons. We have explained the improvement of written bits in Section V-C1. Fewer written bits mean fewer written units, which reduce some write latency. On the other hand, the encoder uses a few of the 8 states which have low latency and energy. For example, in the MLC mode, the encoder selects 4 states with an average latency of 12.75 ns. Compared with the original TLC mode with an average latency of 30.875 ns, that is nearly 2.4 times than that of the MLC mode, writing encoded data in NVM data array can significantly reduce the write latency.

3) Read Latency: Read latency shows the reduction in standby and idle time in the request queue of the NVM module. The average read latency is shown in Figure 19. We can observe that EDFPC reduces the read latency by about 19% compared with DFPC. The reduction of write latency accelerates the processing of instructions. The reduction in written bits does not have a significant impact on read latency, since the decoder has to read all the data in the cache line to decode the written data into the original data. With the combination of the decoding tag bit and the compression prefix, the decoder can quickly determine whether the data needs to be decoded.

4) IPC Improvement: The average IPC is shown in Figure 21. We can observe that EDFPC speeds up the whole TLC-

NVM-based main memory system by about 17% compared with DFPC. In the previous evaluation, EDFPC is effective in reducing the write and read latency. Compared with IPC, which reflects the overall performance of the system, the latency is only part of a measure of system performance. Therefore, the improvement of IPC is a little lower than that of the latency.

5) Energy Improvement: As with the the evaluation of write latency, we evaluate the energy consumption with the configuration in Figure 1. The average total energy is shown in Figure 22. We can observe that EDFPC reduces the energy consumption by about 20% compared with DFPC. In the MLC mode, the encoder selects 4 states with an average energy of 4.68 pJ. Compared with the original TLC mode with an average energy of 16.04 pJ, that is nearly 3.4 times than that of the MLC mode, writing encoded data in NVM data array can significantly reduce the energy consumption.

VI. Related Work

A. Write Reduction

There are many schemes aiming to reduce the amount of write-bits and improve the write performance. Data Comparison Write (DCW) [35] exploits the non-volatility of NVMs and checks the new write-bits with existing bits to determine the modified bits. Only the modified bits require a write operation. DCW can reduce the write power consumption to a half. Flip-N-Write (FNW) [25] compares new data with existing data to reduce the number of bit-flips. FNW reduces the number of bits changed more than half by using only one tag bit per comparing unit. 2-Stage-Write [42] was proposed to increase parallelism within a bank by leveraging the time and power asymmetry of writing '0' and '1'. 2-Stage-Write decreases the number of '1' in a write access by flipping a word if the number of '1' is more than half of the whole word. The performance of 2-Stage-Write mainly depends on the new data without extra read overhead. Moreover, DeWrite [43] proposes a lightweight deduplication scheme to eliminate entire-line duplicate writes to encrypted NVMs.

Compressions are also widely used in NVM-based main memory systems. Kong and Zhou [44] try to improve the lifetime and privacy of NVM. They study redundant bit write removal (RBR) in NVM-based main memory system and observe that encryption with counter-mode can make some previously proposed wear-leveling techniques inefficient. To solve this problem, they propose an encryption scheme to dynamically adjust the strength of ECC and reduce the amount of write-bits since data encryption techniques introduce randomness and reduce data redundancy. Yang et al. [45] propose a cache compression, named Frequent Value Compression (FVC), using several tags to replace some values with high frequency of occurrence. This scheme only catches a few values and has limited performance in floating point benchmarks. Alameldeen and Wood [26] observe that most data stored in fixed size only use several bits and they propose a significance-base compression scheme, call sFPC. Dgien et al. [27] combine sFPC with FNW, and observe that DCW and FNW can not work as expected with the compressed data. They then propose an incremental FPC with a wear-leveling policy to gain performance improvement with an additional position tag bit. They also demonstrate the improvement compared with FVC (combined with FNW). Base-delta-immediate Compression [37] (BDI) exploits the value locality of the words in one cache line. BDI has high compression ratio, but delivers inefficient performance in some application when combined with FNW. Palangappa and Mohanram [38] propose a coding scheme in MLC/TLC NVMs, which encodes the contents of cache lines with static integer patterns and BDI patterns. Hycomp [46] is a hybrid cache compression framework, which selects the appropriate compression based on the feature of the cache lines in Last Level Cache. This general framework has high compression ratio and strong compatibility. Our DFPC can be also implemented within Hycomp.

B. Energy Saving

Due to the asymmetry between read and write operations in NVMs [47] and reliability, performing a write operation requires high current. The high levels of power is a crucial challenge for the NVM devices and the system, especially during the high frequency of write accesses. The power budget also limits the extendibility of the write units and the improvement of parallel. Hay et al. [48] propose a power-token-based scheme, which leverages wear reduction schemes to reduce the write power. The proposed scheme consists of some policies, which focus on monitoring the changed bits during write accesses instead of the whole cache line. By using this scheme, the memory controller can perform more write accesses to reduce the latency. Li et al. [49] design a PCM write scheme and implement with hardware circuits to rearrange all original data units and maximize the power budget utilization, which causes hardware overhead.

C. Endurance Improvement

The poor endurance is also one of the most serious problems in NVM-based main memory systems. Currently, numerous researches focus on improving the reliability and extending the lifetime of NVM cells. P2F [50] is a compression-based scheme to postpone the occurrence of hard errors and improve the lifetime of MLC PCM devices, which uses byte-level compression to convert blocks from MLC to SLC mode. Du et al. [51] study the performance of hybrid DRAM/PCM memory systems. They propose a DRAM cache organization for compression. When performing a write operation. The proposed approach compares the new data with the existing data in NVMs and computes the difference-value (D-value). The D-value is compressed and stored in DRAM due to the low latency of DRAM. The compression can dynamically determine to compress the data or not. The approach only compresses the frequently-modified data and aims to reduce the write operations in NVMs by increasing the write operations in DRAM and the read operations in NVMs. Start-Gap [52] is an effective wear-leveling technique for transforming the logical memory addresses into the physical addresses with two registers. Start-Gap also combine with simple address-space

randomization techniques to significantly enhance the NVM endurance. Fine-Grained Wear-Leveling [11] rotates the cache blocks within the physical NVM frame and reduces the write traffic in NVMs to explore a trade-off for the hybrid system consisting of NVM and DRAM.

VII. Conclusion

NVMs has the advantages of scalability and energy efficiency in the idle status. However, NVMs face some new challenges with the endurance, power and performance. To improve write performance of NVMs and offer the flexibility of compression schemes, we propose a well-adaptive NVM write scheme, called Dynamic Frequent Pattern Compression (DFPC). DFPC compresses the data contents based on a set of specific patterns to minimize the written data. We observe that some zero characters in the specific localities, where the zero-character appears frequently, form a periodic cycle in the typical data distribution. By considering these localities as the compressible part of a word, we extract dynamic patterns by sampling and analyzing the regularity of bit value distribution in the incoming data at runtime. DFPC also extends the data patterns by exploiting the value locality in a cache line to include more kinds of patterns and achieve higher compression ratio. To accelerate the processing of instructions and reduce the total energy consumption in the context of MLC/TLC NVMs, we propose a latency-optimized and energy-efficient DFPC for MLC/TLC NVMs, i.e., EDFPC, to encode the compressed data with low energy and latency states. Experimental results demonstrate that DFPC and EDFPC reduce the amount of write-bits, write latency, and read latency while gaining about 1.2× IPC improvements, compared with the state-of-the-art compression write schemes. While we take PCM as an example of NVMs, the proposed scheme should also improve the performance of other NVMs, such as ReRAM and STT-RAM, which have long write latency and low cell endurance. We have released the source code for public use.

Acknowledgement

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61772212. The preliminary version appears in the Proceedings of the 21st Design Automation and Test in Europe (DATE), 2018.

References

[1] P. J. Nair, D. H. Kim, and M. K. Qureshi, "Archshield:architectural framework for assisting dram scaling by tolerating high error rates," in *Proc. ISCA*, 2013.

[2] Z. Li, R. Zhou, and T. Li, "Exploring high-performance and energy proportional interface for phase change memory systems," in *Proc. HPCA*, 2013.

[3] L. Wilson, "International technology roadmap for semiconductors," *International Technology Roadmap for Semiconductors*, 2011.

[4] H. Zhang, X. Chen, N. Xiao, and F. Liu, "Architecting energy-efficient stt-ram based register file on gpgpus via delta compression," in *Proc. DAC*, 2016.

[5] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase change memory architecture and the quest for scalability," *Communications of the ACM*, vol. 53, no. 7, pp. 99–106, 2010.

[6] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, 2010.

[7] L. Wilson, "International technology roadmap for semiconductors (itrs)," *Semiconductor Industry Association*, 2013.

[8] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung *et al.*, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465–479, 2008.

[9] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 2–13.

[10] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH computer architecture news*, vol. 37, no. 3. ACM, 2009, pp. 14–23.

[11] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[12] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "Fpb: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proc. MICRO*, 2012.

[13] Z. Li, F. Wang, D. Feng, Y. Hua, W. Tong, J. Liu, and X. Liu, "Tetris write: Exploring more write parallelism considering pcm asymmetries," in *Proc. ICPP*, 2016.

[14] 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," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 383–394, 2010.

[15] F. Huang, D. Feng, W. Xia, W. Zhou, Y. Zhang, M. Fu, C. Jiang, and Y. Zhou, "Security rbsg: Protecting phase change memory with security-level adjustable dynamic mapping," in *Proc. IPDPS*, 2016.

[16] L. Jiang, Y. Du, B. Zhao, Y. Zhang, B. R. Childers, and J. Yang, "Hardware-assisted cooperative integration of wear-leveling and salvaging for phase change memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 2, p. 7, 2013.

[17] H. Mao, X. Zhang, G. Sun, and J. Shu, "Protect non-volatile memory from wear-out attack based on timing difference of row buffer hit/miss," in *Proc. DATE*, 2017.

[18] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.

[19] P. Zuo and Y. Hua, "A write-friendly hashing scheme for non-volatile memory systems," in *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST)*, 2017, pp. 1–10.

[20] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell reram memory design," in *Proc. DAC*, 2013.

[21] L. Jiang, Y. Zhang, and J. Yang, "Er: elastic reset for low power and long endurance mlc based phase change memory," in *Proceedings of the ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 39–44.

[22] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Bit mapping for balanced pcm cell programming," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 428–439.

[23] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 40, 2015.

[24] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 153–162.

[25] S. Cho and H. Lee, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," in *Proc. MICRO*, 2009.

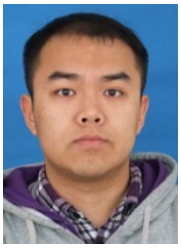
[26] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[27] 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. NANOARCH*, 2014.

[28] Y. Guo, Y. Hua, and P. Zuo, "Dfpc: A dynamic frequent pattern compression scheme in nvm-based main memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*, 2018, pp. 1622–1627.

[29] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5

simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[30] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.
[31] D. Niu, Q. Zou, C. Xu, and Y. Xie, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *Proc. ICCD*, 2013.
[32] M. Zhao, L. Jiang, Y. Zhang, and C. J. Xue, "Slc-enabled wear leveling for mlc pcm considering process variation," in *Proc. DAC*, 2014.
[33] F. Bedeschi, R. Fackenthal, C. Resta, and E. M. Donze, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, 2009.
[34] T. Nirschl, J. B. Philipp, T. D. Happ, and G. W. Burr, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Proc. Intl. Electron Devices Meeting*, 2007.
[35] 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. ISCAS*. IEEE, 2007, pp. 3014–3017.
[36] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *Proc. ISCA*, 2005.
[37] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: practical data compression for on-chip caches," in *Proc. PACT*, 2012.
[38] P. M. Palangappa and K. Mohanram, "Compex++: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvms," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 1, p. 10, 2017.
[39] Z. Li, F. Wang, Y. Hua, W. Tong, J. Liu, Y. Chen, and D. Feng, "Exploiting more parallelism from write operations on pcm," in *Proc. DATE*, 2016.
[40] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. ACM ASPLOS*, 2015.
[41] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *Proc. ACM ASPLOS*, 2016.
[42] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 282–293.
[43] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 442–454.
[44] J. Kong and H. Zhou, "Improving privacy and lifetime of pcm-based main memory," in *Proc. DSN*, 2010.
[45] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *Proc. MICRO*, 2000.
[46] A. Arelakis, F. Dahlgren, and P. Stenstrom, "Hycomp: A hybrid cache compression method for selection of data-type-specific compression methods," in *Proc. MICRO*, 2015.
[47] F. Xia, D. Jiang, J. Xiong, M. Chen, L. Zhang, and N. Sun, "Dwc: Dynamic write consolidation for phase change memory systems," in *Proc. ICS*, 2014.
[48] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing pcm banks from seizing too much power," in *Proc. MICRO*, 2011.
[49] Z. Li, F. Wang, D. Feng, Y. Hua, J. Liu, and W. Tong, "Maxpb: Accelerating pcm write by maximizing the power budget utilization," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 4, p. 46, 2016.
[50] M. Jalili and H. Sarbazi-Azad, "Tolerating more hard errors in mlc pcms using compression," in *Proc. ICCD*, 2016.
[51] Y. Du, M. Zhou, B. Childers, R. Melhem, and D. Mossé, "Delta-compressed caching for overcoming the write bandwidth limitation of hybrid main memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, p. 55, 2013.
[52] 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. MICRO*, 2009.



Yuncheng Guo received the BE and Master degrees in computer science and technology from Huazhong University of Science and Technology (HUST), China, respectively in 2015 and 2018. His research interests include non-volatile memory, algorithms of hashing and data analytics.

Yu Hua received the BE and PhD degrees in computer science from the Wuhan University, China, in 2001 and 2005, respectively. He is a professor at the Huazhong University of Science and Technology, China. His research interests include file systems, cloud storage systems, non-volatile memory, big data analytics, etc. He has more than 100 papers to his credit in major journals and international conferences including IEEE Transactions on Computers (TC), IEEE Transactions on Parallel and Distributed Systems (TPDS), OSDI, MICRO, USENIX FAST, USENIX ATC, ACM SoCC, SC, HPDC, ICDCS, IPDPS, MSST. He serves for multiple international conferences, including ASPLOS, SOSP, USENIX ATC, ICS, RTSS, SoCC, ICDCS, INFOCOM, IPDPS, DAC, MSST, DATE. He is the distinguished member of CCF, senior member of IEEE and ACM, and the member of USENIX.

Pengfei Zuo received the BE degree in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 2014. He is currently a PhD student majoring in computer science and technology at HUST. His current research interests include data deduplication, non-volatile memory, and key-value store. He publishes several papers in major conferences including OSDI, MICRO, USENIX ATC, SoCC, ICDCS, IPDPS, MSST, DATE, etc. He is a student member of IEEE.