Approximate Similarity-Aware Compression for Non-Volatile Main Memory

Zhang-Yu Chen[1] (陈章玉), Yu Hua[1,*] (华宇), *Distinguished Member, CCF, Senior Member, ACM, IEEE*, Peng-Fei Zuo[1] (左鹏飞), Yuan-Yuan Sun[1] (孙园园), and Yun-Cheng Guo[1] (郭云程)

[1] *Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

E-mail: {chenzy, csyhua, pfzuo, sunyuanyuan, ycguo}@hust.edu.cn

Abstract Image bitmaps, i.e., data containing pixels and visual perception, have been widely used in emerging applications for pixel operations while consuming lots of memory space and energy. Compared with legacy DRAM, non-volatile memories (NVMs) are suitable for bitmap storage due to the salient features of high density and intrinsic durability. However, writing NVMs suffers from higher energy consumption and latency compared with read accesses. Existing precise or approximate compression schemes in NVM controllers show limited performance for bitmaps due to the irregular data patterns and variance in bitmaps. We observe the pixel-level similarity when writing bitmaps due to the analogous contents in adjacent pixels. By exploiting the pixel-level similarity, we propose SimCom, an approximate similarity-aware compression scheme in the NVM module controller, to efficiently compress data for each write access on-the-fly. The idea behind SimCom is to compress continuous similar words into the pairs of base words with runs. The storage costs for small runs are further mitigated by reusing the least significant bits of base words. SimCom adaptively selects an appropriate compression mode for various bitmap formats, thus achieving an efficient trade-off between quality and memory performance. We implement SimCom on GEM5/zsim with NVMain and evaluate the performance with real-world image/video workloads. Our results demonstrate the efficacy and efficiency of our SimCom with an efficient quality-performance trade-off.

Keywords approximate computing, data compression, memory architecture, non-volatile memory.

1 Introduction

Many emerging applications, e.g., image/video processing, computer vision, and machine learning, operate on pixels, which are maintained as raw images, called image bitmaps, and stored in main memory for fast accesses by offsets [1]. However, the storage of bitmaps demands a large amount of memory and energy in DRAM. Conventional software-based image compression schemes (e.g., JPEG [2]) are not applicable, since these image-based applications need to access raw images for computation. For example, the kernel in the sobel algorithm [3] is used to read and modify the pixels one by one. Compressed images are still required to be restored into bitmaps on memory for application uses.

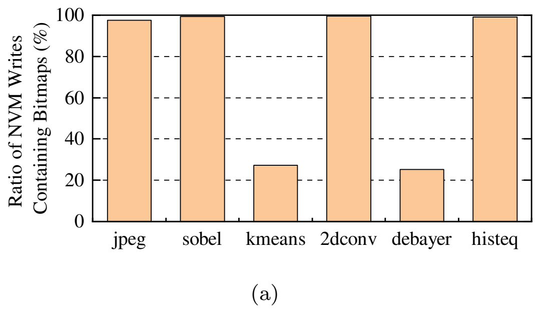
Unlike conventional DRAM, Non-Volatile Memories (NVMs), such as Phase Change Memory (PCM) [4, 5] and Resistive RAM (ReRAM) [6], avoid frequent refresh operations and activation power while providing high density, which are suitable for emerging applications involving bitmaps. NVMs offer DRAM-scale read latency and power, but the required power for writes is much higher than that of DRAM. Due to the maximal current constraint during programing, the write size is limited. Therefore, NVMs suffer from high write power and latency [7–12].

Regular Paper

A preliminary version of this paper was published in the Proceedings of DAC 2020.

This work was supported in part by National Natural Science Foundation of China under Grant No. 62125202 and U22B2022.

*Corresponding Author



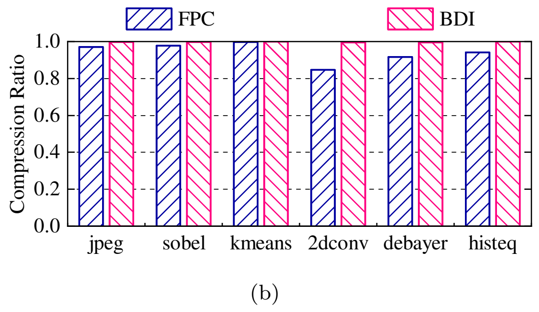


Fig.1. Performance using typical compression schemes on image-based applications. (a) Percentage of NVM writes containing image bitmaps. (b) Compression ratio using FPC and BDI.

Recent designs propose data compression based on general-purpose patterns inside the NVM module controller to reduce bit writes and improve memory performance[13–16]. Specifically, upon receiving an NVM write request, these hardware-layer compression schemes partition the data into words, which are compressed using general-purpose data patterns, e.g., frequent patterns in Frequent Pattern Compression (FPC)[16] and base words with small deltas in Base Delta Immediate (BDI)[15]. After compression, small compressed data are written to NVMs, thus improving the write efficiency. However, for write accesses of bitmaps, the partitioned words are hard to match the general data patterns due to the large variance, which results in high compression ratios (compressed data size relative to uncompressed data size). In order to verify the poor efficiency in existing compression schemes, we record the percentage of NVM writes containing bitmaps (Fig.1(a)) and corresponding compression ratios for these writes (Fig.1(b)) in six image-based workloads (Subsection 5.1). Results in Fig.1 indicate that the writes of bitmaps account for a large portion of NVM writes for image-based applications. However, the average compression ratios of FPC and BDI are 94.2% and 99.8%, which means most data writes of image bitmaps obtain poor compression performance and even become incompressible using precise compression schemes.

Recent research explored the approximate storage for data that tolerate minor inaccuracies[1, 17, 18]. The approximate image storage proposed by Guo et al.[17] leverages the significant entropy differences in the encoded bits of compressed images and applies different levels of error correction codes for different bits. However, the entropy differences are negligible in bitmaps, since each bit in bitmaps corresponds to at most one pixel. Recent work[1, 18] exploits the inter-block similarity (the block here denotes CPU cache blocks) to provide approximate storage for bitmaps. However, searching for similar data in NVMs during each write access incurs extra latency and hardware overheads. Since a large portion of data to be written are approximable (Fig.1(a)), it is possible to improve the write performance by approximately compressing the data on-the-fly before writing to NVMs. In order to efficiently reduce the bit-writes of bitmaps in NVM systems, there are two challenges for data compression.

Irregular Data Patterns. The data in NVM writes containing bitmaps are hard to match the general data patterns in existing compression schemes. Bitmaps consist of the bits in each pixel, and a typical pixel consists of three bytes. Since the pixel size in common bitmaps (e.g., 3 B) is not the same as the word size in conventional compression schemes (e.g., 4 B), there is significant variance in partitioned words. Besides, the value of each word depends on the contents of bitmaps. Therefore, the partitioned words in conventional schemes show irregular data patterns, leading to poor compression performance.

Bitmap Format Variance. When multiple applications (or threads) are running on top of NVM systems with different bitmap formats (e.g., color/grayscale images and different bits per pixel), write accesses to NVMs contain different data layouts. Moreover, the persistence order is determined by the cache replacement policy, which is different from the program order[11, 19]. Due to the reordering, it is challenging to determine the bitmap format for each write access. Data compression designed for one bitmap may fail in others due to the significant changes in data patterns.

Existing bidirectional precision scaling[20] partitions data using annotated word size and conducts approximate precision scaling for error-tolerant data to reduce the data size. Specifically, it approximately encodes the Most Significant Bits (MSBs) and truncates Least Significant Bits (LSBs) of error-tolerant data within the accuracy constraint. However, the pixel value in bitmaps is often stored using the smallest data type, in which identical MSBs are usually unavailable in bitmaps. Moreover, indiscriminately truncating LSBs reduces the color depth and causes noticeable quality degradation.

To address the above two challenges, we propose SimCom, an efficient hardware-level similarity-aware compression scheme, to reduce the bit writes of bitmaps into NVMs, thus improving the memory performance of NVMs. For the first challenge, we leverage the pixel-level similarity in bitmaps and only write a base word (the representative word for a group of continuous similar words) with a run (the number of words in the group) for each group of continuous similar words, which eliminates the writes of similar words in NVMs. The storage costs for small runs are optimized by reusing the LSBs of base words without significant accuracy loss. For the second challenge, SimCom executes compression modes in parallel and adaptively selects an efficient compression mode without programmer annotations on image/video formats.

Compared with the preliminary version of SimCom[21], this paper makes the following main improvements:

- We add important details on the motivation, background, and design to make this paper self-contained. For example, we evaluate the ratios of NVM writes containing approximable data in six workloads to show the opportunities for approximate compression.
- We add several new experiments to demonstrate the efficiency of SimCom and analyze the trade-offs in approximate compression. Different memory architectures (e.g., DRAM-only and DRAM/NVM hybrid main memory) are evaluated to study the energy efficiency of SimCom. Instead of two fixed output error constraints (3% and 5%) in the previous work, we evaluate more configurations to show the quality-performance trade-off in SimCom. The breakdown of the bit-write reduction on NVM by using SimCom is presented and analyzed.
- We discuss the compression modes, overheads, and the architecture support for SimCom and summarize the related work.

Overall, we make the following contributions in SimCom:

- *Similarity-aware Compression.* We develop a model to quantify the pixel-level similarity. With the model, we propose an efficient approximate data compression scheme in hardware layer to reduce the bit-writes of image bitmaps in NVMs on-the-fly.
- *Adaptiveness for Different Formats.* With the domain knowledge of bitmaps, we propose an adaptive scheme to perform approximate compression without prior knowledge about data formats, thus eliminating the annotations on the data types and the pixel size of bitmaps.
- *System Implementation.* We have implemented the prototype of SimCom on GEM5/zsim with NVMain and conducted experiments with real-world workloads in various domains. Results using image/video based applications show that SimCom achieves average 18.3%/22.2%/21.1% energy savings and 17.3%/24.9%/28.8% write latency reduction over FPC/BDI/BiScaling with 3% quality loss.

2 Background and Motivation

2.1 Image Bitmap

Structure Organization. An image bitmap is a pixel storage structure containing the bits for each pixel color. The bits of a pixel color consist of multiple primary colors. The values of one primary color for all pixels comprise a channel. A typical bitmap consists of three channels (e.g., red, green, and blue). For each pixel, the number of bits per channel is eight. Some bitmaps contain an optional channel, called alpha channel, to store transparency information[22, 23]. We use channel count (CC) to represent the number of channels in a bitmap and bits per channel (BPC) to denote the number of bits per channel for each pixel.

Quality Metric. Root-mean-square error (RMSE) is an objective metric to measure the quality of an image, which indicates the difference of each pixel compared with a baseline image. The RMSE of image $\hat{y}$ with respect to baseline image y is calculated using (1), where m denotes the number of pixels in each image. The value of RMSE ranges from 0 to 1 and the lower value is better performance. We use RMSE to measure the output quality of relaxed images like prior work[3, 20].

$$RMSE = \sqrt{\frac{1}{m}\sum\nolimits_{i=0}^{m-1}(y_i - \hat{y}_i)^2}. \tag{1}$$

2.2 Bit-write Reduction in NVMs

To address the high energy consumption in write operations, bit-write reduction techniques are widely used in NVM-based main memory[10, 14–16, 24–28]. Related schemes include data encoding[24, 25, 29], data compression[15, 16, 27], and their combinations[14, 26, 28]. Before writing data into NVMs, compression schemes decrease the data size. For read accesses, the compressed data are decompressed. Data encoding schemes are used to reduce the bit flips in write operations. Moreover, encoding technologies can be leveraged to encode the compressed data for further energy efficiency[14] and lifetime improvement[26].

2.3 Approximate Storage

Approximate storage leverages the error-tolerance of approximable data to slightly relax the accuracy constraints for improvement in terms of performance, data density, lifetime, and energy efficiency. Approximable data are interpreted as the data tolerating minor inaccuracies. In the context of this paper, approximable data denote image bitmaps. For approximate storage, typical approximation consists of three steps: identification of approximable data, approximate techniques, and quality control. Before execution, error-tolerant data should be separated from raw application data, which is accomplished by programmer annotations[18, 30–34] and domain knowledge[17, 35]. For error-tolerant data, traditional guarantees for accuracy in the storage systems are relaxed for gains in memory performance and efficiency. Existing approximate techniques include decreasing refresh rate[30] and lowering voltage[36] in DRAM, using worn blocks and skipping program-and-verify iterations in Multi-Level Cell (MLC) PCM[37], associating similar cache blocks with the same tag entry[18, 33], and utilizing selective error correction code[17, 35]. Given accuracy constraints, we need to select appropriate approximation parameters[1, 33] to achieve suitable trade-off between output quality and performance. The parameters can be inferred dynamically by monitoring the intermediate results[38, 39], using the input features[40], and tuning with canary inputs[41, 42].

2.4 Pixel-level Similarity

Pixel-level similarity is interpreted as the similarity among words in the data of an NVM write access. As shown in Fig.2, the contents of adjacent pixels A, B, C, and D are similar. Instead of fixed four-byte word size, the data in SimCom are partitioned at the pixel-level granularity, e.g., three bytes for RGB format (more details are available in Subsection 3.4). In a bitmap, each pixel describes the color of a tiny point of the image. Hence, adjacent pixels tend to have similar contents. For the storage of an image bitmap, the contents usually are mapped to a continuous region in memory and have continuous addresses in the address space. When a write access of bitmap is issued to NVM module and we partition the data at the boundaries of pixels, partitioned words are possible to be similar due to the analogous contents in adjacent pixels. This paper proposes to leverage the pixel-level similarity in data for approximate compression, thus reducing the data size and improving the memory performance.

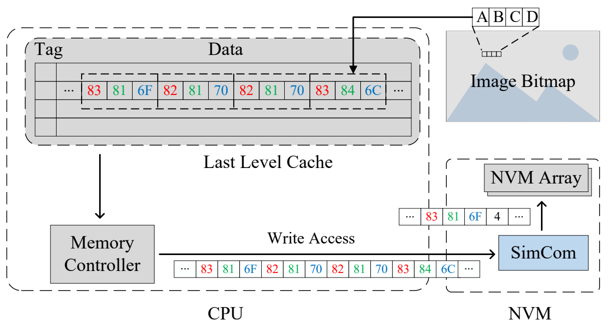


Fig.2. An example of leveraging pixel-level similarity to compress data writes.

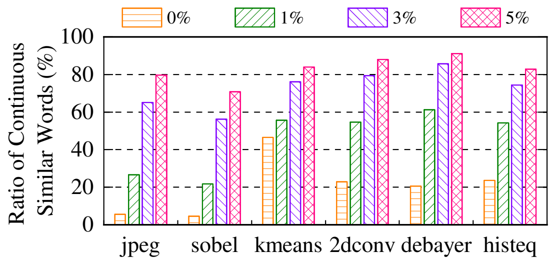


Fig.3. Ratio of continuous similar words in approximable data with different error thresholds. The ratio is interpreted as the size of all continuous similar words divided by total approximable data in bytes.

We have conducted experiments to verify the prevalence of pixel-level similarity in write accesses to NVMs by recording continuous similar words in approximable data, i.e., data containing image bitmaps. Continuous similar words are interpreted as a group of sequential words, in which any two words are similar. We use the proposed model in Subsection 3.3 to quantify the similarity among words. Approximable data are partitioned at the pixel boundaries. Error thresholds denote the normalized difference (Subsection 3.3) and range from 0% (precise) to 100% (maximal approximation), which indicates the approximation degree. The details of experimental settings are described in Subsection 5.1. Fig.3 shows the percentage of continuous similar words in approximable data with different error thresholds. When we increase the error threshold, the ratio of continuous similar words increases up to 82.8% on average.

The pixel-level similarity is common in bitmaps due to two reasons: (1) The changes among adjacent pixels are generally slight. For example, most backgrounds in images consist of similar colors and lack of abrupt changes. (2) The resolution of images is high. With higher resolution for advanced sensors and application requirements, the number of pixels corresponding to one item increases and the difference between two adjacent pixels decreases. The common similarity of pixels offers the opportunity for approximate compression.

Note that even when the error threshold is 0%, the ratio of continuous similar words is still more than 4.5% and up to 46.5%. The substantial similarity in images motivates us to exploit the pixel-level similarity for bit-write reduction and energy efficiency in NVMs.

3 Similarity-aware Data Compression

3.1 Design Overview

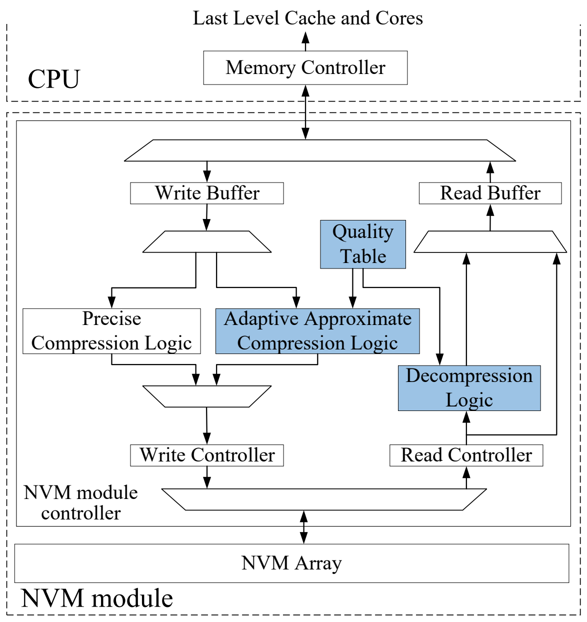


Fig.4. Architecture overview of SimCom.

Fig.4 shows the hardware architecture overview of SimCom. Specifically, Adaptive Approximate Compression Logic and Decompression Logic implement the compression and decompression schemes of SimCom, respectively. Quality Table is an on-chip cache[1, 34], which stores some user-annotated metadata (i.e., start and end addresses, error thresholds) about bitmaps. Quality Table contains only a few entries (e.g., 64) so that the hardware overheads are negligible.

Write Operation. For approximable data containing bitmaps (indicated by Quality Table), Adaptive Approximate Compression Logic partitions data into words, finds continuous similar words, and compresses them into base words and runs. By leveraging the pixel-level similarity, SimCom efficiently reduces the size of approximable data. For precise data, i.e., data not covered by Quality Table, Precise Compression Logic leverages existing precise compression schemes (e.g., FPC[16]) to ensure the correctness, thus enabling the simultaneous executions of various applications.

Read Operation. For NVM read accesses, compressed data are decompressed in Decompression Logic. With the encoded mode index in the compressed data, the decompression logic restores the approximable data for read requests.

3.2 Software Interface

A software interface, i.e., setApproxRegion(sAddr, eAddr, TH), is leveraged to deliver user annotated approximable data regions ([sAddr, eAddr]) and error thresholds of normalized difference (TH) to Quality Table of NVM module controllers via memory-mapped registers[20]. If the addresses of an NVM write request are not overlapping with any regions stored in Quality Table (physical memory addresses of [sAddr, eAddr]), corresponding data are processed via Adaptive Approximate Compression Logic. A practical way to determine the threshold is to search a suitable value using small canary inputs and apply the threshold on full-size inputs[41, 42].

3.3 Similarity Model

Although pixel-level similarity exists in bitmaps, how to efficiently detect the similarity remains a problem. We develop a model to estimate and quantify the pixel-level similarity between two words. The model is based on the observation that corresponding primary colors are similar if two words correspond to similar pixels. Hence, we use the maximal absolute difference in different channels to quantify the similarity between two words. The difference is normalized to the maximal value of primary color, called $maxValue$. The normalized difference between words p and q is calculated using (2).

$$normDiff = \frac{max\{|p[i] - q[i]|\}}{maxValue}, i \in [0, cc). \quad (2)$$

In (2), the $p[i]$ and $q[i]$ correspond to primary colors in the same channel. $maxValue$ is a constant determined by the number of bits for one primary color, i.e., BPC. When BPC is 8, $maxValue$ is 255. cc denotes the value of CC. If one of the two words is a partial word, cc is substituted by the number of common channels in the two words. When the normalized difference is smaller than the error threshold annotated by users, two words are similar.

3.4 Data Partition

Data partition is the first step for compression and nontrivial. According to the proposed similarity model, we need to partition the data at the pixel boundaries in order to find out the similar words. However, how to identify pixel boundaries in data becomes a problem. We cannot figure out the positions of pixel boundaries without additional context information, such as the offset of the data in bitmaps and the corresponding bitmap format. A straightforward solution is to allocate each pixel with a fixed alignment (the alignment should be a factor of the cache block size, e.g., 4 B), thus enabling static pixel boundary positions in the memory space. However, when the actual pixel size (e.g., 3 B) mismatches the alignment, the unused space (e.g., 1 B) in each pixel significantly increases the NVM storage overhead.

In order to preserve the similarity in partitioned words with low overheads, we propose a uniform scheme to partition the data in a write access. Due to the pixel-level similarity, the data form an approximate periodic cycle of the pixel size. Therefore, we propose to partition at the granularity of pixel size and leave the possible remaining bytes (when the data size is not a multiple of the pixel size) at the end as a partial word. For example, if the pixel size is 3 bytes and the data write size is 64 bytes, data are divided into 21 words of 3 bytes and 1 partial word of 1 byte.

3.5 Searching Continuous Similar Words

Since continuous similar words require that any two words are similar (Subsection 2.4), the time complexity to obtain a group of exact continuous similar words is $O(n^2)$ (n denotes the number of words). The high time complexity incurs high latency and hardware overhead to accurately find all continuous similar words in a write access.

In order to alleviate the cost of searching similar words during compression, we propose to approximately search for continuous similar words. Specifically, we slightly relaxed the requirements of continuous similar words. The words in relaxed continuous similar words are only required to be similar to the base word (for simplicity, we use continuous similar words to represent relaxed continuous similar words in the following text unless specified). Even in the relaxed similarity model, the maximal normalized difference in a group is constrained to two times of the annotated error threshold.

Though the appropriate candidate for a base word is the average of all similar words, we take the first word of each group as the base word for two reasons (in the following text, we use base word and base interchangeably): (1) taking the first word as the base simplifies the compression logic; (2) despite the selection of the first word as a base, the penalty in compression ratio is slight.

With the relaxation in similarity and selection of the base for continuous similar words, the time complexity of getting continuous similar words decreases to $O(n)$, which efficiently decreases the complexity of compression logic and improves the compression performance.

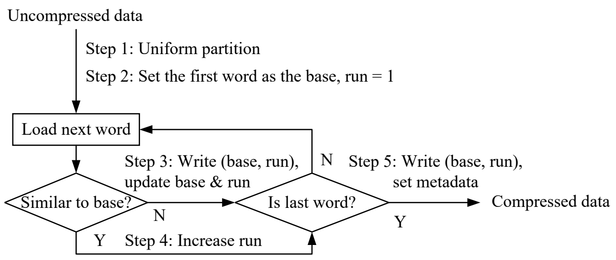

Uncompressed data
Step 1: Uniform partition
Step 2: Set the first word as the base, run = 1
Load next word
Step 3: Write (base, run), update base & run
N
Step 5: Write (base, run), set metadata
Similar to base?
Is last word?
Compressed data
N
Y
Y
Step 4: Increase run

Fig.5. An approximate compression workflow in SimCom.

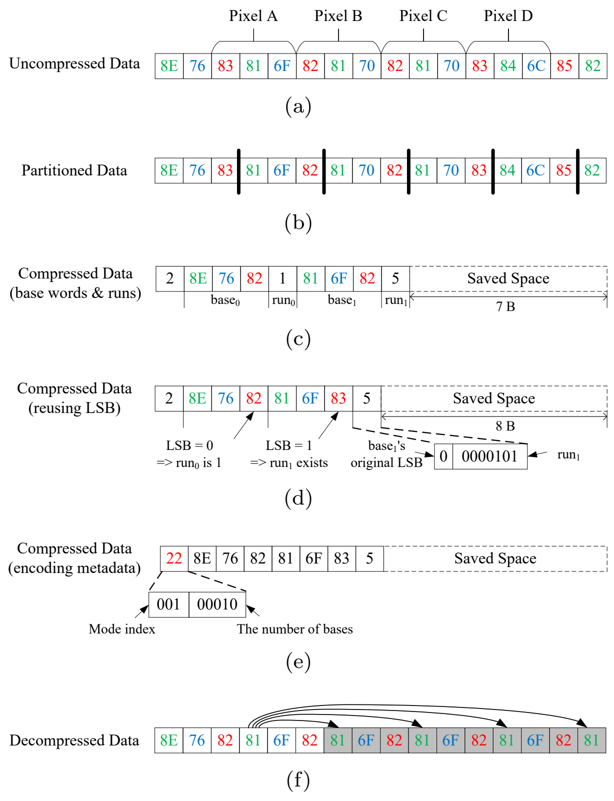


Fig.6. An example of approximate compression and decompression when the error threshold for normalized difference is 0.05. (a) Original data before compression. (b) Data partition. (c) Compressed data using base words and runs. (d) Compressed data after reusing LSB. (e) Compressed data after encoding metadata. (f) Uncompressed data after decompression.

3.6 Similarity-aware Compression for NVMs

Compression for NVM writes. Fig.5 illustrates the workflow of approximate compression. After partitioning the original data (Fig.6(a)) into words as shown in Fig.6(b) (step 1), SimCom sets the first word as the base and initializes the run to 1 (step 2). If the loaded word is not similar to the base, current values of base and run are encoded and written into compressed data. Current word is set as the new base and the run is reset to 1 (step 3). If current word is similar to existing base, SimCom only increases the run by one (step 4). Note that for the possible partial word at the end of data to be written, SimCom obtains the normalized difference between the partial word and the last base via the proposed similarity model (Subsection 3.3). After processing each word, SimCom records the last pair of base and run in the compressed data. The first byte of compressed data is used to record the number of bases and the mode index (step 5). Fig.6(c) shows the result of approximate compression using base words and runs. If the data are not approximable, the data are compressed using existing precise data compression schemes, e.g., FPC.

The above approximate compression workflow is a variant of run-length encoding: the base word is a representative word for near-duplicate words and the run denotes the repeated time. The compression scheme is efficient for data when the run is large, since multiple similar words are replaced by a base word with a run. However, the storage overheads for runs become high when the runs are small. For example, due to the first run in Fig.6(c), the compressed data for the first partitioned word is one byte larger than the original word. To mitigate the metadata overheads of small runs, we propose to reuse the LSBs of base words to encode small runs. Specifically, the LSB of a base word is used to indicate whether its corresponding run exists in the compressed data. If the LSB of the base word is 1, the run exists in compressed data and we further reuse the MSB of the run to store the original LSB of its corresponding base word; otherwise, the run is 1 and not stored. For example, in Fig.6(d), since run_1 (i.e., 5) is larger than 1, the original LSB of base_1 is stored in the MSB of run_1, avoiding affecting the accuracy of bases that are similar to other words. The LSB of base_0 is 0 and indicates that run_0 is 1. As a result, only LSBs of words not similar to adjacent ones are affected and the accuracy loss is limited. According to our evaluation results, the worst quality degradation of reusing LSB (i.e., indiscriminately truncating the LSB of all base words) is 1.35% when BPC is 8. Reusing 2 bits leads to nonnegligible quality loss of 2.62%, since the typical output quality constraint for images is 3%[20]. Hence, SimCom only reuses the LSB of a base word.

Decompression for NVM reads. For read accesses to approximately compressed data, the approximate decompression is used to reconstruct the stored data. Specifically, for each pair of base and run in compressed data, the base is used to fill the read buffer multiple times according to the run. Fig.6(f) shows an example of the decompression. Essentially, the bases are used to represent similar words (i.e., the shadowed bytes in the figure). For precise compressed data, the encoded data are reconstructed by the inverse procedure of precise compression (e.g., FPC). If the loaded data are not compressed, the data would bypass the decompression logic and respond to read accesses.

4 Adaptive Approximate Compression

In order to handle different bitmap formats, the approximate compression proposed in Section 3 requires extra metadata including CC and BPC. Though it is possible to annotate the metadata to be stored in cache tags[33] or an address table in memory controllers[20, 34], these techniques cause additional overheads and programmer annotations. Moreover, users need to confirm the bitmap formats and annotate these metadata before execution. Hence, in this section, we propose to leverage the image characteristics and adaptively select the appropriate mode for data compression without additional programmer annotations.

4.1 Adaptive Compression Scheme

The proposed scheme selects from predefined compression modes in an adaptive manner.

1) Why Use Predefined Compression Modes for Different Image Formats? The images generally include grayscale and color images. Grayscale images contain only one channel and the color images in RGB color space consist of red, green, blue, and optional alpha channels. In other color spaces (e.g., YUV), similar components (e.g., one luminance channel and two chrominance channels) exist. The BPC in common images is 8 bits, which represent 256 levels in each channel. The 24 bits per pixel (3C1B) represent more than 16 million colors, while the number of colors discriminated by the human eye is up to 10 million[43, 44]. For applications processing HD (high-definition) images, 16 bits per channel is enough to encode the necessary colors. Therefore, we propose to use six compression modes to handle different image formats. The options for CC are 1 (e.g., grayscale), 3 (e.g., RGB), and 4 (e.g., RGB with an alpha channel) and the options for BPC are 8 and 16 by default.

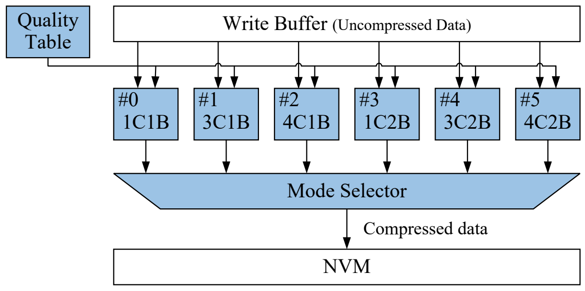


Fig.7. Adaptive compression scheme overview. The two integers in each compression mode denote the number of channels and the bytes per channel.

2) How to Determine the Suitable Compression Mode for a Write Access? A straightforward approach is to sample some write accesses for an efficient compression mode to be applied on later NVM writes. Sampling works when all write accesses have regular pattern formats (e.g., all applications using one bitmap format). However, sampling often fails when data writes have random pattern formats (e.g., applications using different bitmap formats are running in an NVM system). Instead of sampling, SimCom performs six compression modes in parallel and selects the compression mode with the minimal mean difference. As shown in (3), mean difference is calculated as the average difference between every two adjacent words in data. We observe that the mean difference of the right compression mode (i.e., the mode matching the bitmap format) is minimal, which makes sense due to the pixel-level similarity. Fig.7 shows the overview of adaptive compression scheme used in SimCom. Six compression modes with different CCs and BPCs process data in parallel. *Mode Selector* first selects the mode with the minimal mean difference. If multiple modes have the minimal mean difference, the mode selector chooses the one with the minimal compressed data size. For the simplicity of compression logic, SimCom reuses the normalized difference between each word and the base as the difference between adjacent words. Due to the error-tolerance of application and the similarity between words and their bases, the reuse of normalized difference is acceptable (evaluated in Section 5).

$$meanDiff = \frac{1}{n}\sum\nolimits_{i=0}^{n-1} normDiff(w[i], w[base(i)]). \tag{3}$$

4.2 Metadata Management

There are two classes of metadata in SimCom. The first-class metadata are used for approximately compressed data in SimCom including the choice of compression mode and the number of bases. Except for the 1C1B compression mode, each base occupies at least 2 bytes. Therefore, there are no more than 32 bases in compressed data with 64-byte write data granularity. The number of bases can be encoded using 3 bits. Hence, for all compression modes except for 1C1B, SimCom uses the first byte of the compressed data to encode the choice of compression mode (the highest 3 bits) and the number of bases (the rest 5 bits). Fig.6(e) shows an example of compacting the mode index of 3C1B and the number of bases into one byte. For the 1C1B compression mode, SimCom stores the number of bases into the second byte of compressed data. The second-class metadata are used for approximate compression including one compressible bit to indicate whether a data block is compressed or not. SimCom stores the compressible bits in a separate region in NVMs like prior work[15, 16, 20]. The compressible bit can be packed into compressed data to reduce NVM accesses and improve memory bandwidth[13, 45].

5 Evaluation

5.1 Experimental Setup

Table 1. System configurations

Component	Configuration	
CPU	Core	One x86-64 core, 2 GHz
	L1 I/D cache	32 KB, 2 ways, LRU
	L2 cache	1024 KB, 8 ways, LRU
	Cache block size	64 B
Memory	Model	PCM
	Controller	FCFRFS
	Read/Write latency	120 ns/150 ns
	Organization	4 GB, 8 B write unit size

We implement SimCom in GEM5[46] with NVMain[47]. For energy comparisons with DRAM based memory systems (Subsection 5.2), we use the energy model of Micron DDR3-1333_4Gb_8B_x8 provided by NVMain. Since SimCom focuses on data compression and is orthogonal to the underlying memory model, we use a First Ready First Come First Serve (FRFCFS) memory controller to serve NVM accesses due to the simplicity and ease of use. The system configurations of GEM5 and NVMain are listed in Table 1. Due to the limited simulation speed of GEM5, we also leverage zsim[48], a fast pin-based x86-64 simulator, to evaluate the video-based applications. We measure the performance with six image-based workloads, i.e., jpeg, sobel, and kmeans from AxBench[3] and 2dconv, debayer, and histeq from PERFECT[49] and one video-based workload, i.e., x264 from PARSEC[50]. These workloads are selected for various domains, i.e., jpeg for compression, sobel, 2dconv, debayer, and histeq for image processing, kmeans for machine learning, and x264 for video processing. Two color spaces are used in the tested workloads (YUV for x264 and RGB for others). The ratios of approximable data in these workloads are shown in Fig.1(a). As suggested in related work[3], we use RMSE (Subsection 2.1) as the metric to measure the output error (i.e., the quality of output image) compared with the precise compression result. Structural similarity (SSIM)[51] is used to quantify the output error of videos by 1-SSIM. The input images come from Kodak dataset①. The output errors are reported using the average RMSE of 6 images. The typical output error constraints are 3% following BiScaling[20] and 5% for more aggressive approximation. Before running these workloads, we warm up the system with 100 million instructions.

We have evaluated the following compression schemes (FNW[25] is used to further reduce bit-flips in all schemes):

- *FPC.* By exploiting the general frequent patterns, FPC[16] compresses the matched words with short prefix bits. For fair comparisons, we enhance this scheme by adding approximation. Specifically, for a partitioned word, if a similar word derived from the word by flipping few bits matches a data pattern, the pattern is used to compress the word.

- *BDI.* This scheme[15] leverages the narrow value characteristics of array and compresses cache block data into bases with small deltas. This scheme is an approximate version of BDI[15]. It relaxes the narrow value constraints and compresses the words that slightly overflow the delta limit.

- *BiScaling.* This scheme[20] uses bidirectional precision scaling to approximately compress the data to be written.

- *ApproxCom.* This is our proposed scheme that leverages the pixel-level similarity for approximate compression. ApproxCom requires annotations on BPC and CC.

- *SimCom.* This is our proposed scheme leveraging pixel-level similarity and adaptive compression (i.e., ApproxCom + adaptive compression), which eliminates the annotations on data formats used in BiScaling and ApproxCom.

Since BiScaling, ApproxCom, and SimCom focus on approximate compression on approximable data, we use precise FPC to compress precise data in these schemes.

①http://r0k.us/graphics/kodak/, June 2022.

Table 2. Annotation requirements in compression schemes

Scheme	Error Threshold	Channel Count	Bits Per Channel
FPC	✓	✗	✗
BDI	✓	✗	✗
BiScaling	✓	✗	✓
ApproxCom	✓	✓	✓
SimCom	✓	✗	✗

Note: ✓: requires the annotation; ✗: does not require the annotation

We leverage programmer annotations[31, 32] and ISA extensions[36] to deliver necessary information into storage systems like prior work[18, 33, 34, 36]. Programmer annotations are mature techniques and widely used in approximate storage systems[18, 31, 33, 34, 36]. We use programmer annotations to annotate bitmaps as approximable data in workloads. Through ISA extensions, write accesses with approximable data are identified and processed by approximate compression logics. Table 2 shows the required annotations for all compression schemes.

For fairness, we tune the approximation degrees in different schemes (e.g., the number of truncated bits for BiScaling, normalized difference thresholds for ApproxCom/SimCom) to achieve the same output error constraints and compare the memory performance.

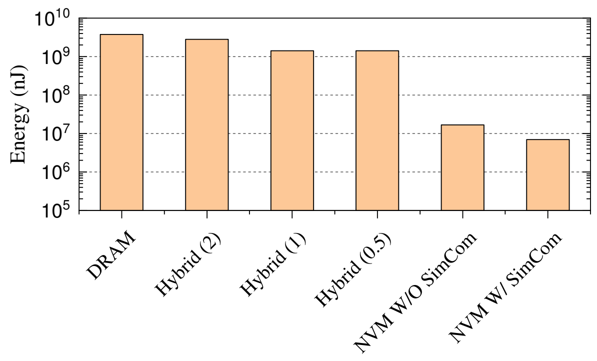


Fig. 8. Energy consumption for different memory systems with 4 GB total capacity using jpeg workload. "Hybrid" indicates a hybrid main memory consisting of DRAM and NVM. The number after "Hybrid" denotes the capacity of DRAM in GB.

5.2 Energy Efficiency

Fig.8 shows the total energy for different memory systems when executing the jpeg workload (other workloads show similar trends). Due to the non-volatility characteristics, NVMs have 225× higher energy efficiency than legacy DRAM. SimCom decreases the energy consumption in NVM by 58.6% with 3% quality loss. For hybrid memory systems of DRAM and NVM, hot pages are stored in DRAM[47]. Hence, bitmaps are stored in DRAM as a cache for NVMs to reduce the writes in NVMs. Although a small DRAM chip in hybrid memory systems reduces the refresh energy, the total energy consumption is still much higher than NVM-only memory systems. The reason is that NVMs completely avoid the expensive refresh operations, thus significantly improving the energy efficiency.

5.3 Quality-Performance Trade-off

5.3.1 *Image-based Workloads*

Figs.9-14 show the memory performance improvement in terms of bit-write ratio, write latency, and energy consumption using different data compression schemes under various output error constraints. During our experiments, we observe serious quality degeneration when the output error approaches to 10%. Therefore, we only plot the curves with output errors under 10%.

Bit-write Ratio. For Figs.9-14, (a)s show the bit-write reduction for image-based workloads with different output error constraints. The bit-write ratio denotes the percentage of bits written on NVM after compression and FNW. A lower bit-write ratio implies a higher NVM performance improvement. With the increase of output errors, the bit-write ratios in all approximate compression schemes decrease. Due to the efficiency of pixel-level similarity, ApproxCom and SimCom often generate fewer bit-writes than other approximate compression schemes with the same output error. SimCom achieves 35.4%/39.6%/34.4% lower bit-write ratios on average than FPC/BDI/BiScaling with the same output error of 3% (the same constraint following related work[20]). When the output error increases to 5%, the average reductions of bit-write ratios become 42.4%/47.0%/40.6%. In kmeans and debayer, the benefits of approximation decrease due to the smaller ratios of approximable data than other workloads, as shown in Fig.1(a). Due to the flexibility of adaptive compression, SimCom obtains slightly lower bit-write ra-

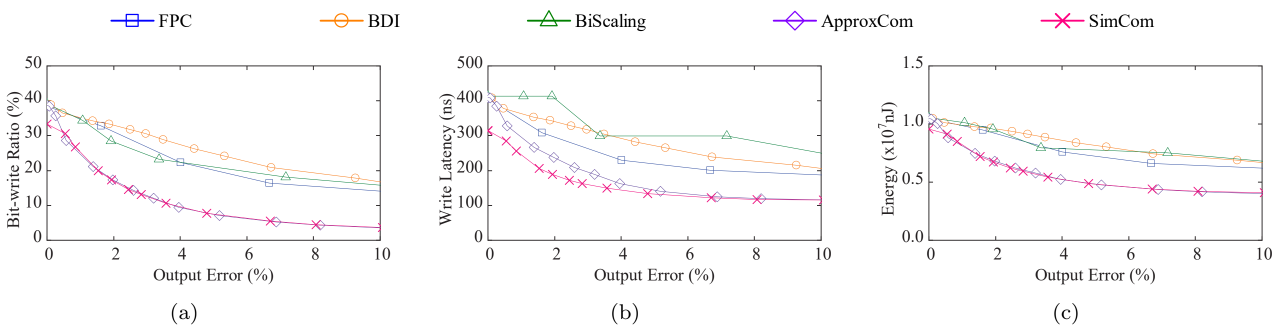


Fig.9. Performance of jpeg. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

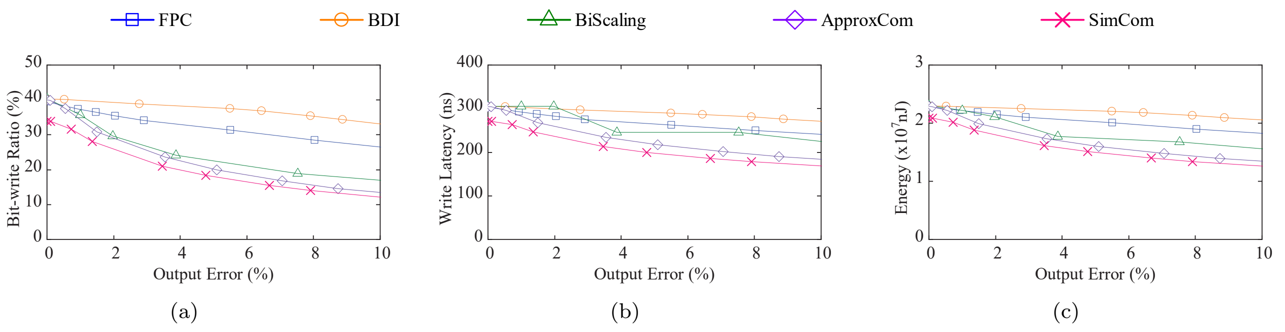


Fig.10. Performance of sobel. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

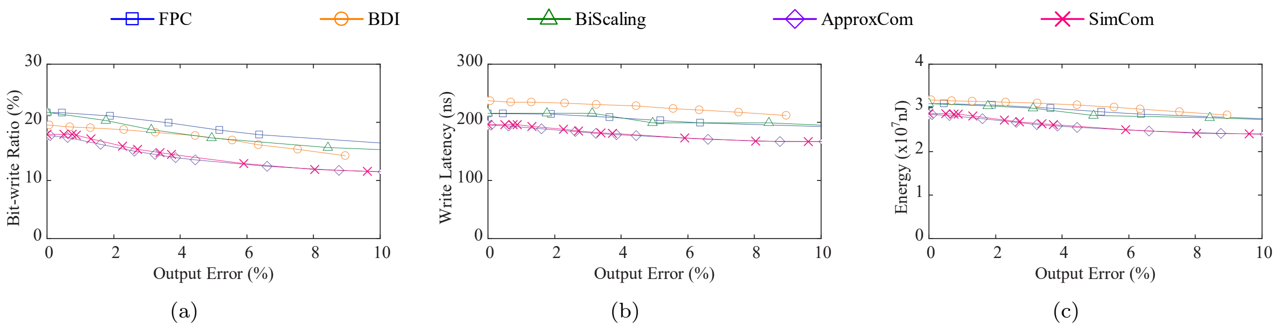


Fig.11. Performance of kmeans. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

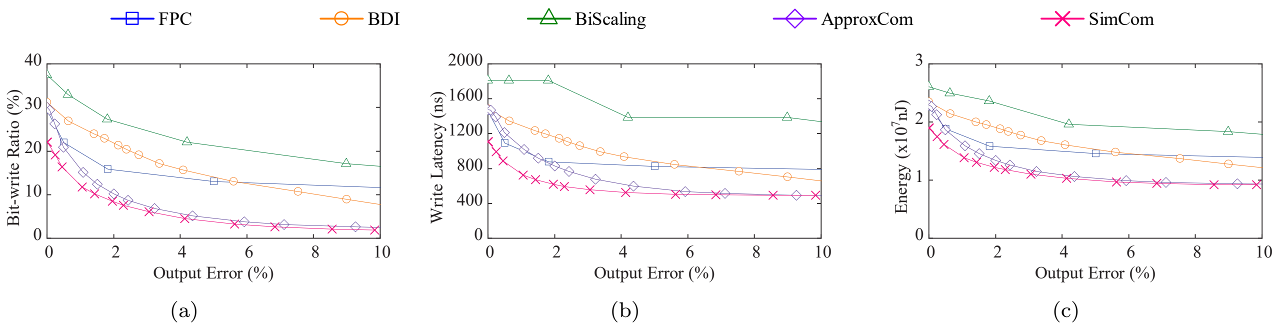


Fig.12. Performance of 2dconv. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

tios than ApproxCom. The reason is that the adaptive compression scheme in SimCom optimizes the compression of grayscale images which are stored in color image formats. For example, if a grayscale pixel is stored in the RGB color space, the bits for each channel are the same. When the mean differences generated by 1C1B

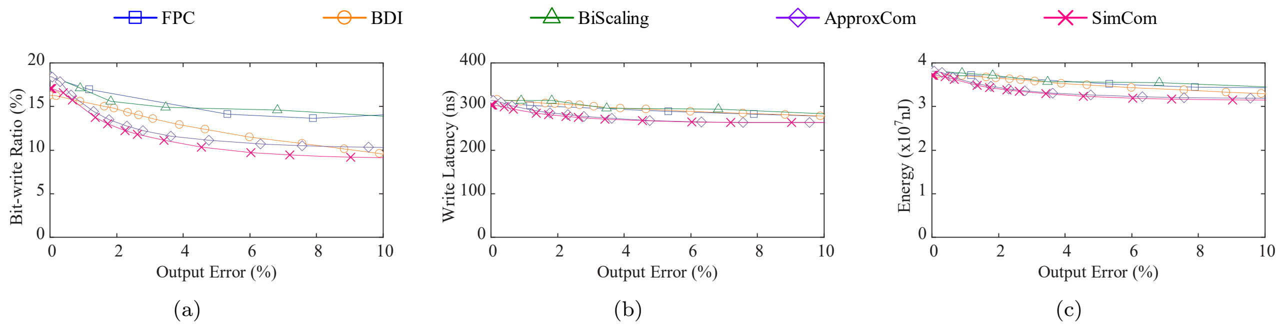


Fig.13. Performance of debayer. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

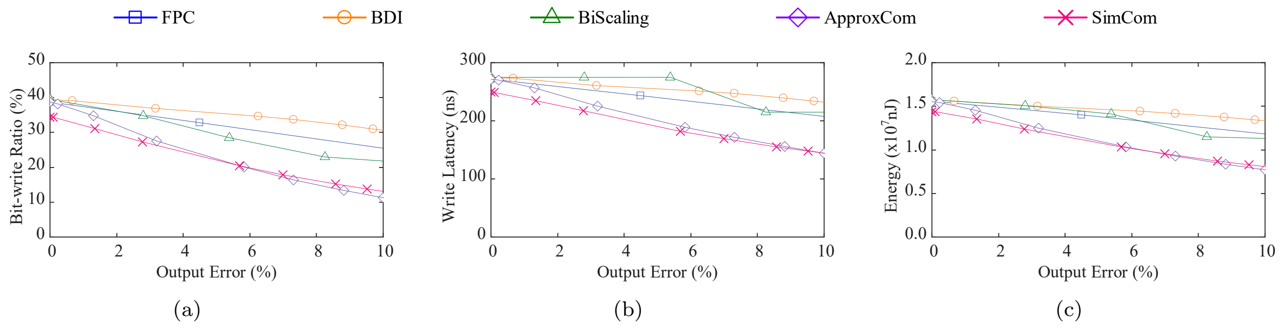


Fig.14. Performance of histeq. (a) Bit-write ratio. (b) Write latency. (c) Energy consumption.

and 3C1B are identical (e.g., 0), 1C1B obtains smaller compressed data size than 3C1B. SimCom prefers the modes with small compressed data sizes (e.g., 1C1B), thus leading to more bit-write reduction than ApproxCom (e.g., 3C1B).

Write Latency. Due to the electric current constraint in NVMs, the write operation is divided into several sequential write units[10, 25]. Therefore, the write latency mainly depends on the data size to be written. As shown in (b)s of Figs.9-14, the trends of write latency in image-based workloads are analogous to the trends of bit-write ratios in these workloads. The superiority of SimCom in terms of bit-write ratio due to the pixel-level similarity turns into the benefits in write latency. Under 3% and 5% quality loss, SimCom respectively achieves on average 19.6%/26.3%/30.0% and 21.8%/28.2%/31.0% write latency reduction compared with FPC/BDI/BiScaling for these six workloads.

Energy Consumption. For Figs.9-14, (c)s show the energy consumption with various quality loss for image-based workloads. Since the energy consumed in the programming process is the main fraction in the energy consumption of NVMs[14], the number of bit-writes determines the energy consumption. Compared with approximate compression schemes, SimCom reduces the consumed energy by 18.3%/22.2%/21.1% and 21.4%/25.6%/23.3% than FPC/BDI/BiScaling with 3% and 5% quality loss, respectively.

5.3.2 *Video-based Workload*

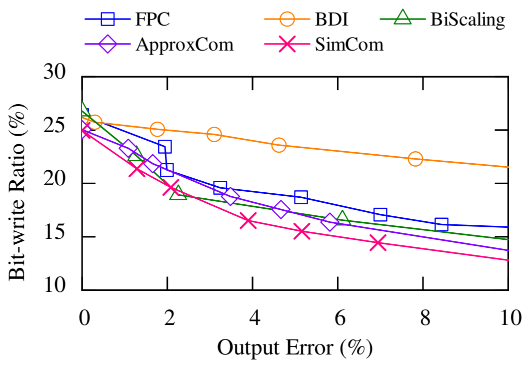


Fig.15. The bit-write ratio in the x264 workload.

Fig.15 shows the bit-write ratio on NVM in the x264 workload, a video processing application. Since videos consist of frames, which are often stored as bitmaps to be processed. By leveraging the pixel-level similarity, SimCom achieves higher bit-write efficiency than other schemes. Under 3% and 5% quality loss for the x264 workload, SimCom respectively shows 9.2%/26.8%/2.2% and 16.7%/33.0%/9.2% lower bit-write ratios than FPC/BDI/BiScaling. For the write latency, SimCom reduces the latency by 15.2%/18.2%/22.1% compared with FPC/BDI/BiScaling under 3% quality loss. We skip the measurement of the energy consumption in the x264 workload, since zsim does not support energy modeling.

5.4 Breakdown of Bit-write Reduction

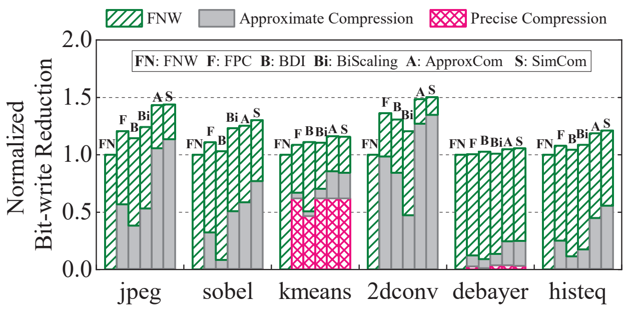


Fig.16. Breakdown of bit-write reduction with 3% output error.

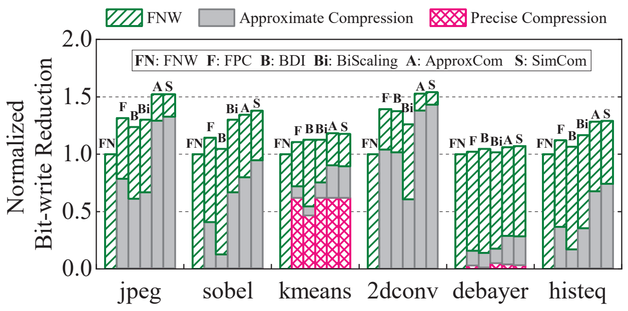


Fig.17. Breakdown of bit-write reduction with 5% output error.

In order to evaluate the contribution of different techniques (i.e., precise compression for precise data, approximate compression for approximable data, and FNW for compressed data) in all evaluated schemes, we record the bit-write reduction from each technique.

As shown in Fig.16 and Fig.17, SimCom gains significant bit-write reduction from the approximate compression for error-tolerant data, thus obtaining more bit-write reduction than other schemes. Though the percentages of precise data are large in kmeans and debayer (Fig.1(a)), the precise compression performance is limited due to the pattern mismatch and irregular data types in these workloads, thus resulting in the inefficiency of precise compression schemes (i.e., FPC and BDI).

5.5 Adaptability for Bitmap Format Variance

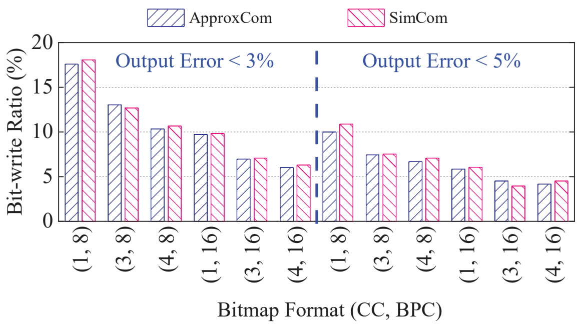


Fig.18. The bit-write ratio in the jpeg workload with different bitmap formats.

In order to verify the adaptability of SimCom, we evaluate the jpeg workload with input images of different formats. As shown in Fig.18, SimCom achieves comparable (within 1%) bit-write ratios to those of ApproxCom. Without annotations on bitmap formats, SimCom is able to infer the data types according to the mean difference (Subsection 4.1) among data. The pixel-level similarity in data guarantees that the right compression mode (Section 4) tends to obtain the minimal mean difference.

An interesting point is that SimCom obtains slightly lower bit-write ratios than ApproxCom when CC is 3, e.g., bitmap formats of (3, 8) and (3, 16) when the output error threshold is 3% and 5%, respectively. In order to investigate the performance improvement, we record the selection of mode inside the mode selector of SimCom. Table 3 shows SimCom is able to obtain the right compression mode in most cases (the numbers in boldface). However, when CC is 3, the mode selector possibly chooses the mode in which CC is 1. The reason is that when the values of three channels in a pixel are identical, e.g., pixels of white color and grayscale bitmaps stored in RGB formats, a compression mode with one channel can achieve the same mean difference with smaller compressed data size than the right compression mode. Therefore, SimCom achieves the adaptiveness in the mode selection and low bit-write ratios for various bitmap formats.

Table 3. Statistics for the selected modes in SimCom under 3% output error constraint

Mode	Bitmap Formats (CC, BPC)					
	(1, 8)	(3, 8)	(4, 8)	(1, 16)	(3, 16)	(4, 16)
1C1B	**82.4**	47.2	0.6	0.2	0.2	0.2
3C1B	0.2	**34.1**	0	0	0	0
4C1B	0.1	0.4	**96.9**	0	0	0
1C2B	15.3	7.4	0	**98.5**	58.3	1.0
3C2B	0	7.1	0	0.2	**38.5**	0
4C2B	0.1	0.1	2.2	0.6	1.6	**97.9**
Incompressible	1.9	3.7	0.3	0.5	1.4	0.9

Since the mode selector selects the compression mode with minimal mean difference, it is possible to select a compression mode with slightly smaller mean difference but much larger compressed data size than the matched compression mode. As shown in Fig.18, the conservative strategy used in SimCom may cause slightly larger compressed data sizes and bit-write ratios than those of ApproxCom.

5.6 Discussion

Approximate Compression Modes. There are six available compression modes in SimCom by default. Compression modes with other CCs and BPCs can be added into SimCom like existing modes. For images with BPC > 16, an alternative approach is to downscale the precision to fit the images for predefined compression modes in SimCom. Due to the error-tolerance in images, slight precision downscaling of images with large BPC would not cause significant quality loss.

Additional Latency of SimCom. Since six approximate compression modes are executed in parallel, the additional compression latency of SimCom is determined by the slowest compression mode. To determine whether a word is similar to a base word, we just need to check if the largest difference in each channel is larger than $TH * maxValue$ or not (Subsection 3.3). According to prior study[15], the additional latency for compression involving calculating differences and comparing results is a few cycles. Although there may be multiple rounds of finding similar words due to possible multiple base words in the 64-byte data, the number of iterations is usually small, since the pixel-level similarity is prevalent in image-based applications as shown in Fig. 3. For decompression, restoring data from base words is simple intra-block copying and fast.

Hardware Overhead of SimCom. The majority of hardware overhead of SimCom comes from the parallel execution using six approximate compression logics, which can be optimized by reusing the logic. In this case, six compression modes are executed one by one in the compression logic, which trades the compression speed for the hardware efficiency.

Architecture Support for SimCom. In the current testbed, SimCom requires architecture support (i.e., microarchitecture modifications) like prior work[18, 33, 34, 36] to identify write accesses with approximable data, which are widely-used techniques in approximate storage systems[18, 31–34, 36]. For image-based applications (e.g., machine learning and computer vision), memory performance and energy efficiency are important for the overall system performance. In addition, these applications are generally tolerant for minor errors. Moreover, power consumption is constrained in specific platforms (e.g., smartphones and embedded devices). Therefore, it is meaningful to provide architecture support for approximate storage systems. With architecture support like Truffle[36], SimCom only requires small hardware changes in the NVM module controller to deliver accuracy requirements via software interfaces without ISA extensions[1, 20, 30]. Through the interfaces, approximable data are stored in a separate memory region. Hence, read or write accesses to the region can be identified by memory addresses.

6 Related Work

Data Compression in NVMs. FPC[16] uses static data patterns to compress frequent patterns into short prefix bits. BDI[15] leverages the characteristics of narrow values in arrays to encode each word using bases with small deltas. However, as shown in our evaluation (Section 5), general-purpose patterns are difficult to match bitmaps even with approximation. Different from FPC and BDI, SimCom leverages the pixel-level similarity in bitmaps and efficiently trades slight output quality for performance improvement. CompEx[14] and CompEx++[52] applies expansion coding to integrate data compression and coding for performance improvements in MLC NVMs. COE[26] leverages the saved space in data compression to store the tag bits of data encoding. These combination schemes are orthogonal to data compression and can be used to further improve the energy efficiency of SimCom.

Approximate Image Storage. To address the challenge of massive image collections, several approximation approaches are proposed to improve the efficiency of image storage. A biased MLC write scheme[17, 35] is used to balance the drift and write errors in MLC PCM. Selective ECC is applied on images according to the importance of encoded bits[17]. Progressively encoding scheme can improve the read performance of images[53]. However, these schemes are established based on the significant entropy differences in encoded image bits, which do not exist in bitmaps. Therefore, encoded image approximation is inefficient for the writes of bitmaps in NVMs. Recent work[1] proposes to selectively write pixels in approximate window by writing soft bits in MLC STT-MRAM main memory. The approximation is efficient when loading entire images from disks to MLC STT-MRAM. However, this technique is specific to MLC STT-MRAM and needs searching for similar contents in other memory blocks, which leverages inter-block similarity and leads to additional hardware overheads and latency when writing data from cache to NVMs.

Approximate Cache and Main Memory. For regions containing error-tolerant data, Flikker[30] reduces the refresh rates of DRAM to improve the energy efficiency. Bidirectional precision scaling[20] is proposed to compress the data to be written to DRAM. However, indiscriminately reducing the precision of all data can significantly decrease the image quality. STAxCache[34] proposes approximate reads/writes tailored for STT-MRAM based L2 cache. Associating similar data blocks with the same cache tags leads to cache performance improvement[33]. The accesses to memory can be served with predicted values according to previous data patterns[18, 54]. The inter-block similarity used in the above caches is orthogonal to the pixel-level similarity of our work. Unlike them, SimCom exploits the pixel-level similarity in bitmaps and efficiently decreases NVM write overheads by approximate compression.

7 Conclusion

SimCom leverages the pixel-level similarity in bitmaps to efficiently reduce the writes of similar words for NVM-based main memory, thus improving the memory performance in terms of energy efficiency and write latency. By exploiting adaptive approximate compression, SimCom mitigates the programmer annotations used for compression. Experiments show that compared with state-of-the-art FPC/BDI/BiScaling, SimCom decreases 18.3%/22.2%/21.1% energy and 17.3%/24.9%/28.8% write latency with slight quality loss of 3%.

SimCom is possible to be extended to leverage the inter-block similarity in approximable data from various applications, which is our future work.

References

[1] Zhao H, Xue L, Chi P, and Zhao J. Approximate Image Storage with Multi-level Cell STT-MRAM Main Memory. In *Proc. the IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2017, pp.268–275. DOI: 10.1109/ICCAD.2017.8203788.

[2] Wallace G K. The JPEG Still Picture Compression Standard. *Commun. ACM*, 1991, 34(4): 30–44. DOI: 10.1145/103085.103089.

[3] Yazdanbakhsh A, Mahajan D, Esmaeilzadeh H, and Lotfi-Kamran P. AxBench: A Multiplatform Benchmark Suite for Approximate Computing. *IEEE Design & Test*, 2017, 34(2): 60–68. DOI: 10.1109/MDAT.2016.2630270.

[4] Xia F, Jiang D, Xiong J, and Sun N. A Survey of Phase Change Memory Systems. *J. Comput. Sci. Technol.*, 2015, 30(1): 121–144. DOI: 10.1007/s11390-015-1509-2.

[5] Wong H P, Raoux S, Kim S, Liang J, Reifenberg J P, Rajendran B, Asheghi M, and Goodson K E. Phase Change Memory. *Proc. IEEE*, 2010, 98(12): 2201–2227. DOI: 10.1109/JPROC.2010.2070050.

[6] Wong H P, Lee H, Yu S, Chen Y, Wu Y, Chen P, Lee B, Chen F T, and Tsai M. Metal-oxide RRAM. *Proc. IEEE*, 2012, 100(6): 1951–1970. DOI: 10.1109/JPROC.2012.2190369

[7] Liu H, Chen D, Jin H, Liao X, He B, Hu K, and Zhang Y. A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions. *J. Comput. Sci. Technol.*, 2021, 36(1): 4–32. DOI: 10.1007/s11390-020-0780-z.

[8] Bittman D, Long D D E, Alvaro P, and Miller E L. Optimizing Systems for Byte-Addressable NVM by Reducing Bit Flipping. In *Proc. the 17th USENIX Conference on File and Storage Technologies*, Feb. 2019, pp.17–30.

[9] Li Z, Zhou R, and Li T. Exploring High-Performance and Energy Proportional Interface for Phase Change Memory Systems. In *Proc. the 19th IEEE International Symposium on High Performance Computer Architecture*, Feb. 2013, pp.210–221. DOI: 10.1109/HPCA.2013.6522320.

[10] Yue J and Zhu Y. Accelerating Write by Exploiting PCM Asymmetries. In *Proc. the 19th IEEE International Symposium on High Performance Computer Architecture*, Feb. 2013, pp.282–293. DOI: 10.1109/HPCA.2013.6522326.

[11] Zuo P, Hua Y, and Wu J. Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory. In *Proc. the 13th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2018, pp.461–476.

[12] Xu J, Zhang L, Memaripour A, Gangadharaiah A, Borase A, Silva T B D, Swanson S, and Rudoff A. NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System. In *Proc. the 26th Symposium on Operating Systems Principles*, Oct. 2017, pp.478–496. DOI: 10.1145/3132747.3132761.

[13] Hong S, Nair P J, Abali B, Buyuktosunoglu A, Kim K, and Healy M B. Attaché: Towards ideal memory compression by mitigating metadata bandwidth overheads. In *Proc. the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2018, pp.326–338. DOI: 10.1109/MICRO.2018.00034.

[14] Palangappa P M and Mohanram K. CompEx: Compression-Expansion Coding for Energy, Latency, and Lifetime Improvements in MLC/TLC NVM. In *Proc. the 22nd IEEE International Symposium on High Performance Computer Architecture*, Mar. 2016, pp.90–101. DOI: 10.1109/HPCA.2016.7446056.

[15] Pekhimenko G, Seshadri V, Mutlu O, Gibbons P B, Kozuch M A, and Mowry T C. Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches. In *Proc. the International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2012, pp.377–388. DOI: 10.1145/2370816.2370870.

[16] Dgien D B, Palangappa P M, Hunter N A, Li J, and Mohanram K. Compression Architecture for Bit-write Reduction in Non-volatile Memory Technologies. In *Proc. the IEEE/ACM International Symposium on Nanoscale Architectures*, July 2014, pp.51–56. DOI: 10.1109/NANOARCH.2014.6880482.

[17] Guo Q, Strauss K, Ceze L, and Malvar H S. High-Density Image Storage Using Approximate Memory Cells. In *Proc. the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2016, pp.413–426. DOI: 10.1145/2872362.2872413.

[18] Miguel J S, Albericio J, Jerger N D E, and Jaleel A. The Bunker Cache for Spatio-Value Approximation. In *Proc. the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2016, pp.43:1–43:12. DOI: 10.1109/MICRO.2016.7783746.

[19] Shin S, Tirukkovalluri S K, Tuck J, and Solihin Y. Proteus: A Flexible and Fast Software Supported Hardware Logging approachs for NVM. In *Proc. the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2017, pp.178–190. DOI: 10.1145/3123939.3124539.

[20] Ranjan A, Raha A, Raghunathan V, and Raghunathan A. Approximate Memory Compression for Energy-efficiency. In *Proc. the IEEE/ACM International Symposium on Low Power Electronics and Design*, July 2017, pp.1–6. DOI: 10.1109/ISLPED.2017.8009173.

[21] Chen Z, Hua Y, Zuo P, Sun Y, and Guo Y. Reducing Bit Writes in Non-volatile Main Memory by Similarity-aware Compression. In *Proc. the 57th ACM/IEEE Design Automation Conference*, July 2020. DOI: 10.1109/DAC18072.2020.9218683.

[22] Porter T K and Duff T. Compositing Digital Images. In *Proc. the 11th Annual Conference on Computer Graphics and Interactive Techniques*, July 1984, pp.253–259. DOI: 10.1145/800031.808606.

[23] Duff T. Deep compositing using lie algebras. *ACM Trans. Graph.*, 2017, 36(3): 26:1–26:12. DOI: 10.1145/3023386.

[24] Yang B, Lee J, Kim J, Cho J, Lee S, and Yu B. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *Proc. the International Symposium on Circuits and Systems*, May 2007, pp.3014–3017. DOI: 10.1109/ISCAS.2007.377981.

[25] Cho S and Lee H. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009, pp.347–357. DOI: 10.1145/1669112.1669157.

[26] Xu J, Feng D, Hua Y, Tong W, Liu J, and Li C. Extending the Lifetime of NVMs with Compression. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2018, pp.1604–1609. DOI: 10.23919/DATE.2018.8342271.

[27] Guo Y, Hua Y, and Zuo P. DFPC: A Dynamic Frequent Pattern Compression Scheme in NVM-based Main Memory. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2018, pp.1622–1627. DOI: 10.23919/DATE.2018.8342274.

[28] Palangappa P M and Mohanram K. CASTLE: Compression Architecture for Secure Low Latency, Low Energy, High Endurance NVMs. In *Proc. the 55th Annual Design Automation Conference*, June 2018, pp.87:1–87:6. DOI: 10.1145/3195970.3196007.

[29] Jacobvitz A N, Calderbank A R, and Sorin D J. Coset Coding to Extend the Lifetime of Memory. In *Proc. the 19th IEEE International Symposium on High Performance Computer Architecture*, Feb. 2013, pp.222–233. DOI: 10.1109/HPCA.2013.6522321.

[30] Liu S, Pattabiraman K, Moscibroda T, and Zorn B G. Flikker: Saving DRAM Refresh-power through Critical Data Partitioning. In *Proc. the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2011, pp.213–224. DOI: 10.1145/1950365.1950391.

[31] Sampson A, Dietl W, Fortuna E, Gnanapragasam D, Ceze L, and Grossman D. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Proc. the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2011, pp.164–174. DOI: 10.1145/1993498.1993518.

[32] Sampson A, Baixo A, Ransford B, Moreau T, Yip J, Ceze L, and Oskin M. ACCEPT: A Programmer-Guided Compiler Framework for Practical Approximate Computing. *University of Washington Technical Report UW-CSE-15-01*, 1, 2015.

[33] Miguel J S, Albericio J, Moshovos A, and Jerger N D E. Doppelgänger: A Cache for Approximate Computing. In *Proc. the 48th International Symposium on Microarchitecture*, Dec. 2015, pp.50–61. DOI: 10.1145/2830772.2830790.

[34] Ranjan A, Venkataramani S, Pajouhi Z, Venkatesan R, Roy K, and Raghunathan A. STAxCache: An Approximate, Energy Efficient STT-MRAM Cache. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2017, pp.356–361. DOI: 10.23919/DATE.2017.7927016.

[35] Jevdjic D, Strauss K, Ceze L, and Malvar H S. Approximate Storage of Compressed and Encrypted Videos. In *Proc. the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2017, pp.361–373. DOI: 10.1145/3037697.3037718.

[36] Esmaeilzadeh H, Sampson A, Ceze L, and Burger D. Architecture Support for Disciplined Approximate Programming. In *Proc. the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2012, pp.301–312. DOI: 10.1145/2150976.2151008.

[37] Sampson A, Nelson J, Strauss K, and Ceze L. Approximate Storage in Solid-State Memories. In *Proc. the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2013, pp.25–36. DOI: 10.1145/2540708.2540712.

[38] Baek W and Chilimbi T M. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *Proc. the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2010, pp.198–209. DOI: 10.1145/1806596.1806620.

[39] Samadi M, Jamshidi D A, Lee J, and Mahlke S A. Paraprox: Pattern-Based Approximation for Data Parallel Applications. In *Proc. the 19th Architectural Support for Programming Languages and Operating Systems*, Mar. 2014, pp.35–50. DOI: 10.1145/2541940.2541948.

[40] Sui X, Lenharth A, Fussell D S, and Pingali K. Proactive Control of Approximate Programs. In *Proc. the 21st Architectural Support for Programming Languages and Operating Systems*, Apr. 2016, pp.607–621. DOI: 10.1145/2872362.2872402.

[41] Laurenzano M A, Hill P, Samadi M, Mahlke S A, Mars J, and Tang L. Input Responsiveness: Using Canary Inputs to Dynamically Steer Approximation. In *Proc. the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2016, pp.161–176. DOI: 10.1145/2908080.2908087.

[42] Xu R, Koo J, Kumar R, Bai P, Mitra S, Misailovic S, and Bagchi S. VideoChef: Efficient Approximation for Streaming Video Processing Pipelines. In *Proc. the USENIX Annual Technical Conference*, July 2018, pp.43–56.

[43] Judd D B. *Color in Business, Science and Industry.* Wiley-Interscience, 1975.

[44] Leong J. Number of colors distinguishable by the human eye. *Hypertextbook,(ed.). Wyszecki, Gunter. Color. Chicago: World Book Inc*, 824, 2006.

[45] Young V, Kariyappa S, and Qureshi M K. Enabling Transparent Memory-Compression for Commodity Memory Systems. In *Proc. 25th IEEE International Symposium on High Performance Computer Architecture*, Feb. 2019, pp.570–581. DOI: 10.1109/HPCA.2019.00010.

[46] Binkert N L, Beckmann B M, Black G, Reinhardt S K, Saidi A G, Basu A, Hestness J, Hower D, Krishna T, Sardashti S, Sen R, Sewell K, Altaf M S B, Vaish N, Hill M D, and Wood D A. The gem5 Simulator. *SIGARCH Computer Architecture News*, 2011, 39(2): 1–7. DOI: 10.1145/2024716.2024718.

[47] Poremba M, Zhang T, and Xie Y. NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems. *CAL*, 2015, 14(2): 140–143. DOI: 10.1109/LCA.2015.2402435.

[48] Sánchez D and Kozyrakis C. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. In *Proc. the 40th Annual International Symposium on Computer Architecture*, June 2013, pp.475–486. DOI: 10.1145/2485922.2485963.

[49] Barker K, Benson T, Campbell D, Ediger D, Gioiosa R, Hoisie A, Kerbyson D, Manzano J, Marquez A, Song L, Tallent N R, and Tumeo A. PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual. *Pacific Northwest National Laboratory and Georgia Tech Research Institute*, 2013.

[50] Bienia C, Kumar S, Singh J P, and Li K. The parsec benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton University, January 2008.

[51] Wang Z, Bovik A C, Sheikh H R, and Simoncelli E P. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Processing*, 2004, 13(4): 600–612. DOI: 10.1109/TIP.2003.819861.

[52] Palangappa P M and Mohanram K. CompEx++: Compression-Expansion Coding for Energy, Latency, and Lifetime Improvements in MLC/TLC NVMs. *ACM Trans. Archit. Code Optim.*, 2017, 14(1): 10:1–10:30. DOI: 10.1145/3050440.

[53] Yan E Q, Zhang K, Wang X, Strauss K, and Ceze L. Customizing Progressive JPEG for Efficient Image Storage. In *Proc. the 9th USENIX Workshop on Hot Topics in Storage and File Systems*, July 2017.

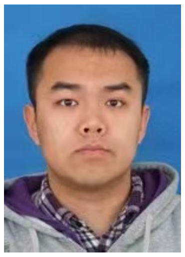
[54] Miguel J S, Badr M, and Jerger N D E. Load Value Approximation. In *Proc. the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2014, pp.127–139. DOI: 10.1109/MICRO.2014.22.

Zhang-Yu Chen received his B.S. degree in computer science and technology from Huazhong University of Science and Technology, Wuhan, in 2017. He is currently a Ph.D. candidate majoring in computer system architecture at Huazhong University of Science and Technology, Wuhan. His research interests include non-volatile memory systems, key-value stores, memory architecture, and debugging.

Yu Hua received his B.S. and Ph.D. degrees in computer science from Wuhan University, Wuhan, in 2001 and 2005, respectively. He is currently a professor at Huazhong University of Science and Technology, Wuhan. His research interests include cloud storage systems, file systems, non-volatile memory architectures, etc. He is a distinguished member of CCF and a senior member of ACM and IEEE.

Peng-Fei Zuo received his B.S. and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology, Wuhan, in 2014 and 2019, respectively. His research interests include memory systems, storage systems and techniques, and security.

Yuan-Yuan Sun received her B.S. and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology, Wuhan, in 2014 and 2019, respectively. Her research interests include cloud storage systems, semantic hashing, metadata management, index structures, and big data analytics.



Yun-Cheng Guo received his B.S. and M.S. degrees in computer science and technology from Huazhong University of Science and Technology, Wuhan, in 2015 and 2018, respectively. His research interests include non-volatile memory, algorithms of hashing, and data analytics.