Improving the energy efficiency of STT-MRAM based approximate cache

Wei Zhao[1], Wei Tong[1*], Dan Feng[1], Jingning Liu[1], Zhangyu Chen[1], Jie Xu[1], Bing Wu[1], Chengning Wang[1], Bo Liu[2]
[1]Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System,
Engineering Research Center of data storage systems and Technology,
(school of Computer Science & Technology, Huazhong University of Science & Technology),
Ministry of Education of China, Wuhan, China
[2]Hikstor Technology Co., LTD, Hangzhou, China
Email:{weiz, tongwei, dfeng, jnliu, chenzy, xujie_dsal, wubin200, chengningwang}@hust.edu.cn,
liubo@hikstor.com

***Abstract*—Approximate computing applications lead to large energy consumption and performance demand for the memory system. However, traditional SRAM based cache cannot satisfy these demands due to high leakage power and limited density. Spin Transfer Torque Magnetic RAM (STT-MRAM) is a promising candidate of cache due to low leakage power and high density. However, STT-MRAM suffers from high write energy. To leverage the ability of tolerating acceptable quality loss via approximations to data, we propose an STT-MRAM based APProximate cache architecture (APPcache) to write/read approximate data thus largely reducing energy. We find many similar elements (e.g. pixels in images) existing in cache lines while running approximate computing applications. Therefore, APPcache uses several light-weight similarity-based encoding schemes to eliminate the similar elements to reduce the data size thus reducing the write energy of STT-MRAM based cache. Besides, we design a software interface to manually control the output quality. APPcache can significantly eliminate similar elements, thus improving energy efficiency. Experimental results show that our scheme can reduce write energy and improve the image raw data compression ratio by 21.9% and 38.0% compared with the state-of-the-art scheme with 1% error rate, respectively. As for the output quality, the losses of all benchmarks are within 5% with 1% error rate.**

***Index Terms*—approximate computing, STT-MRAM, energy**

I. Introduction

In the big data era, approximate computing applications such as machine learning, image processing, are widely used in human's daily life. However, the frequent computations and I/O operations lead to huge pressure for the current computer systems [1]. Consequently, large capacity and high energy-efficiency caches are needed to bridge the performance and energy gap [2]. Unfortunately, the current SRAM based cache faces challenges with the CMOS technology scaling because of increased leakage power and process variations. Spin Transfer Torque Magnetic RAM (STT-MRAM) is a potential cache candidate in recent years due to several advantages, such as high chip density, low leakage power, low read latency, and good compatibility with CMOS [3]. However, the write energy efficiency of STT-MRAM is limited due to the long spin transfer torque switching process, which causes high write energy and long write latency [4].

*Corresponding author: Wei Tong (tongwei@hust.edu.cn)

Emerging applications (e.g., machine learning, image processing) exhibit good intrinsic resilience to data errors, which provides the opportunity to reduce energy through approximation techniques to the underlying data [5]. Approximate computing leverages this property to improve the energy and performance of computing systems at different levels, including software, architecture, and circuits [4] [6]. Several works proposed some schemes to reduce the read and write energy by accessing approximate data [7] [8]. Some other works improve energy efficiency by designing approximate multipliers and adders [9] [10].

To fully use the error resilience of approximate computing applications, recent works proposed some approximate encoding or compression schemes to balance the output quality and energy efficiency. Biscaling [11] truncates the data bits of both the most significant bit (MSB) and least significant bit (LSB) direction. But the MSBs can be truncated only if all MSBs of the elements are the same, which leads to a low compression ratio. For existing approximate applications, such as machine learning, the image raw data are stored in the main memory and cache to access data fast. Because of the similarity of raw data, some schemes are proposed to reduce the data redundancy. AxBA [5] is an approximate bus architecture, reducing the bus traffic by removing redundant data of typical approximate computing applications. However, this scheme uses the first element as the only base, which cannot effectively eliminate redundant elements. Furthermore, SimCom [12] selects several bases to reduce data redundancy for main memory, but it still chooses the first element as the base of each similar part, which is not effective enough. Besides, the tag bits for recording the similar elements may cause large bit writes.

In this work, we propose **APPcache**, an energy-efficient cache architecture for several approximate applications. We observe that the main bit flips written to STT-MRAM based cache are caused by image or video raw data. Besides, raw data shows data similarity in the cache line. We observe that selecting the geometric mean value of the minimum and maximum data in the data set as the base element can reduce more similar elements. Based on the key observation, we propose a similarity-based encoding scheme to reduce the written data size thus improving energy efficiency. Besides, we propose

978-3-9819263-5-4/DATE21/©2021 EDAA

`run_tag` to record similar elements, and we observe that many full-similar (all elements in the cache line are similar with the base) cache lines exist in several benchmarks. Therefore, we use a 1-bit tag to indicate the state of full-similar to avoid large extra writes. Then, for the bases followed by several similar elements, these bases must be written precisely. For the bases without similar elements, we can choose the most energy-efficient value as the base from all possible values. The main contributions of this work are as follows:

- Three key observations: (1) The main bit flips are from the raw data; (2) The geometric mean base can gain more similar elements with the same error rate; (3) Many full-similar cache lines exist in several benchmarks.
- We propose several energy-efficient techniques to reduce the write energy of STT-MRAM based cache.
- We propose APPcache, an energy-efficient cache architecture. Besides, a software interface is designed for programmers to set the output image quality.
- Experimental results show that APPcache can reduce write energy by 21.9% and improve raw data compression ratio by 38% compared with the state-of-the-art scheme, respectively. The quality losses of all benchmarks are within 5% with 1% error rate.

II. Background and Motivation

A. STT-MRAM Basics

An STT-MRAM cell is composed of one transistor and one Magnetic Tunneling Junction (MTJ), i.e. 1T1M structure. MTJ is the main component of STT-MRAM, and different resistance states of MTJ correspond to different logic values (0 or 1) [3]. Fig. 1(a) shows the structure of MTJ. It is composed of two ferromagnetic layers (Free Layer and Reference Layer) and one oxide barrier layer (MgO). The magnetization direction of the reference layer is fixed, and the free layer's magnetization direction can be parallel (P) or anti-parallel (AP) to the reference layer. P and AP indicate the cell is in low resistance state (logical 0) and high resistance state (logical 1), respectively. The transistor can control the current that flows through the bit line (BL) and source line (SL) to write and read data bits. Fig 1(b) shows the structure of SLC STT-MRAM.

While writing the data of an STT-MRAM cell, the write current switches the magnetization direction of the free layer, causing a bit flip. Due to the large current and long switching time, writing data consumes significant energy [4]. For a read operation, a small sensing current is applied to generate a bit line voltage (V_{BL}). This V_{BL} is then compared with a reference voltage to decide whether a logical 1 or a logical 0 is stored in the cell. The details of reading and writing data are diagrammed in Fig. 1(c).

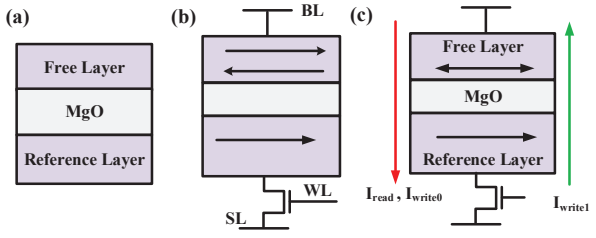

Fig. 1: (a) MTJ structure (b) The structure of SLC STT-MRAM (c) Read and write operations.

B. Approximate Storage

For emerging approximate computing applications, the data (e.g., image, video, media, etc.) can tolerate minor errors. Thus, it is possible to improve the energy efficiency and memory performance at the sacrifice of minor accuracy. Previous works proposed to store the encoded image/video (i.e. .jpeg or .mp4 format) data in MLC PCM substrate storage [13] [14]. However, for approximate computing applications, CPU or GPU uses the raw data of image and video for computation, and the encoded data need to be decoded in the cache and memory. Several schemes proposed to eliminate the number of similar cache lines [15] [16], thus effectively reduce the overall writes. However, searching for similar blocks and maintaining the persistence of cache lines incurs extra hardware overhead and performance loss. MLC STT-MRAM based approximate main memory [17] is used to store images. This work reduces write energy by writing only one of several neighboring similar pixels to the soft domain of MLC STT-MRAM. But this technique wastes the capacity of the hard domain. Biscaling [11] is proposed to compressed data written into the main memory with both the MSB and LSB direction, but this scheme highly depends on the data patterns. SimCom [12] and AxDedup [5] are two effective techniques that harnessing the intra-block data similarity to reduce bit flips. However, they use the first element as the base, which cannot effectively eliminate similar elements. Unlike them, our APPcache can largely reduce the writes to similar elements with little quality loss.

C. Motivation

1): We test the bit flips of several typical approximate computing benchmarks from TABLE II with the system configuration in TABLE I. We find that the main bit flips are from writing the image raw data of approximate computing applications. From the results shown in Fig. 2, we can know the bit flips of image data take up the main part of overall bit flips, thus leading to large energy consumption.

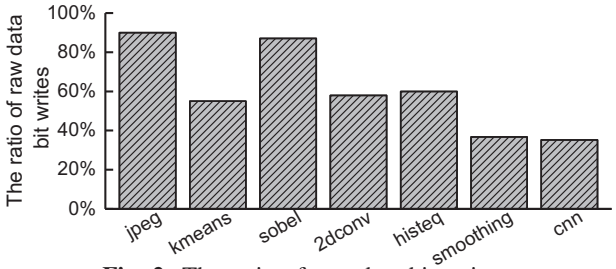

Fig. 2: The ratio of raw data bit writes.

2): Due to the pixel-level similarity of image and the address continuity, the data in cache lines exhibit good similarity. We formulate error threshold th_{app} in Equation 1. E_i and E_{base} are pixel values of the ith and base elements, respectively. While the difference with base is within the error threshold, E_i is identified similar to the base. The relative error rate is defined as Equation 2. max_{value} is the possible maximum value (e.g. 255 for 8-bit image).

$$|E_i - E_{base}| \leq th_{error} \quad (1)$$

$$error\_rate = \frac{th_{error}}{max_{value}} \quad (2)$$

Increasing the th_{error} can relax the data similarity. We find that selecting the first element cannot totally remove similar elements with a specified error rate. For example, there is a image data set $A = \{80, 83, 81, 84, 85, 88\}$. The error threshold th_{error} is set to 4, and the difference of each value in the set is not huge. Selecting the first element leads to only four similar elements. However, we find that difference between the minimum and maximum elements is less than $2 * th_{error}$. Thus we can use the geometric mean (Gmean base) value of the maximum and minimum elements (i.e., 84) as the base. This method leads all elements in the data set are similar. Now, we prove that this method can make all similar elements at the same error threshold. We give a data set that all values satisfy Equation 3.

$$max - min \leq 2 * th_{error} \tag{3}$$

$$Gmean = \frac{max + min}{2} \tag{4}$$

Either min or max has the largest arithmetic difference max_{diff} with Gmean base.

$$max_{diff} = max - Gmean = \frac{max - min}{2} \leq th_{error} \tag{5}$$

or

$$max_{diff} = |min - Gmean| = \frac{|min - max|}{2} \leq th_{error} \tag{6}$$

From Equation 4, 5, 6, we can prove that all data are within the same error threshold.

We test the normalized number of full-similar (all elements in the cache line are similar with the base) cache lines under these two methods. As depicted in Fig. 3, our Gmean base can significantly increase the number of full-similar cache lines, and we can also learn that many cache lines are full-similar.

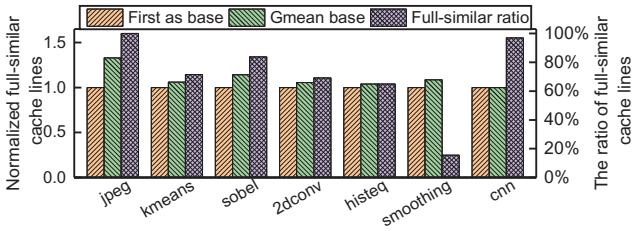

Fig. 3: Normalized full-similar cache lines of two ways.

III. Design

The overview of APPcache architecture is shown in Fig. 4. While a block address hits the image data region, the coming data will be encoded/decoded with our proposed scheme. If not, this block is accessed precisely. Especially, the cache controller cannot identify the physical address of image data. APPcache needs programmers to annotate the virtual address that we want to approximately access [4] [11] [12]. The virtual address can be translated to physical address through Translation Lookaside Buffer (TLB).

A. Similarity Based Encoding Algorithm

Selecting the Base. The cache lines of approximate computing benchmarks exhibit good data similarity, and minor changes of data make little quality loss. We select the geometric mean value of the maximum ($Value_{max}$) and minimum ($Value_{min}$) data as the base to gain more bit flips reduction. Fig. 5 shows the example of choosing the base element. This cache line is

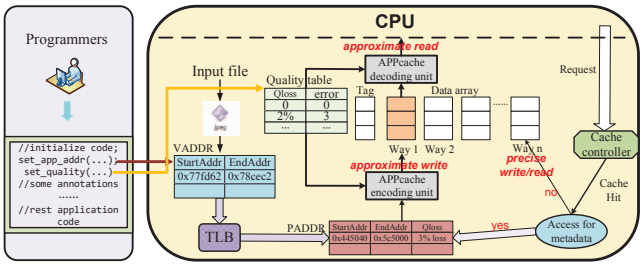

Fig. 4: The overview of APPcache architecture.

derived from *kmeans*. The image format of *kmeans* is RGB. Due to the continuity of similar pixels, we can choose the first three channels as the first element, then every three channels are divided into an element. As for element6, it has one channel. While the three channels (one for element6) of an element are all similar to the base, this element can be identified as similar. We introduce the process of comparing one channel. Firstly, we set the first element as the initial $Value_{max}$ and $Value_{min}$. Then, we compare the $Value_{max}$ and $Value_{min}$ with the new element to determine new $Value_{max}$ and $Value_{min}$. If the new values satisfy Equation 3, the new element can be considered similar. Next, we continue this process until an element cannot satisfy the constraint or reaching the end of a cache line. Then, the base of one channel is computed as the geometric mean value of $Value_{max}$ and $Value_{min}$. The comparison procedures of channels proceed in parallel. When a channel value cannot satisfy Equation 3, the element is not similar, and the encoder will repeat the process above to start a new base. As for the cache line in Fig. 5, all elements are similar, and we only have to write one base thus largely reducing bit writes. Besides, we allocate 4 bits to record the bitmap formats and data types. The details of selecting the base of one channel are shown in Algorithm. 1.

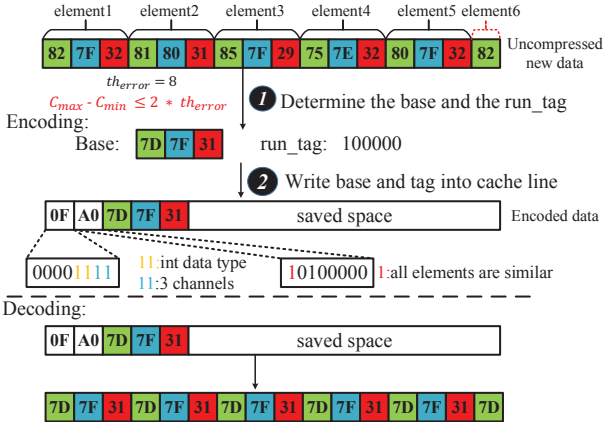

Fig. 5: The example of our similarity based encoding.

Recording the Bases and Similar Elements. To effectively record the elements thus decoding data, we propose run_tag to track the bases and similar elements. For a cache line from *kmeans* as shown in Fig. 5, the run_tag of the first element is set to '1', which means the start location of several similar elements. While the following run_tag is set to '0' if the subsequent element is similar to the base. If one element cannot satisfy the similarity constraint shown in Equation 3,

the corresponding tag bit is set to '1' to start a new base. With this scheme, only 6 bits are required for the cache line of *kmeans* to record the base and similar elements. Besides, due to the existence of many similar elements, many '0' data are in run_tag. In other words, there are many same data ('0') in the old and new tags, leading to low bit flips. Furthermore, as shown in Fig. 3, the full-similar cache lines take up the main part of overall cache lines. Based on this, we set a 1-bit tag to indicate whether this cache line is fully similar or not. With this technique, the writes to run_tag can be replaced by writing the 1-bit tag. While decoding data, the decoder can get the information of bases and similar elements through run_tag. The details of the encoding process are shown in Algorithm. 1.

Algorithm 1: Similarity Based Encoding

Input: C_i (C_i is a channel value of ith element), anotations of data type and image format.

Output: C_{base}, run_tag

$C_{min}, C_{max}, C_{base} = C_0$;
$run\_tag[0] = 1$;
for $i = 1; i \leq pixel_{num}; i++$ **do**
  $C_{min} = min(C_i, C_{min})$;
  $C_{max} = max(C_i, C_{max})$;
  if $C_{max} - C_{min} \leq 2th_{app}$ **then**
    $run\_tag[i] = 0$;
    $C_{base} = (C_{max} + C_{min})/2$;
  else
    $run\_tag[i] = 1$;
    $C_{min}, C_{max}, C_{base} = C_i$;

B. Selecting the Most Energy-efficient Base

After the similarity-based encoding, some bases may have no similar elements. For these bases that have more than 1 similar run, they must be written precisely. While the bases that have no similar elements (i.e. zero runs) can be approximately written. We observe many zero-runs bases existing after the similarity-based encoding. Besides, the value of these bases can be written approximately with several values. To reduce the bit flips, we can choose the base which has the minimum bit flips. Fig. 6 shows an example of selecting the most energy-efficient base. There are two elements in the data after similarity-based encoding. From the run_tag, we can know that the first base has no runs. Then, the encoder calculates bit flips with all optional values to decide the base which produces the minimum bit flips with the old cache line data. For the green channel, the possible value ranges from '0' to '15' while the error threshold is 8. Next, the encoder calculates the bit flips of several possible values with the old data to avoid extra bit writes. Finally, the value 'A' is selected for the green channel. The other channels similarly operate in parallel. When decoding data, the bases with no runs can be directly read out.

C. Software Interface

To minimize the burden of programmers, we propose a software interface that allows programmers to manually select approximation tolerant regions in the system address space so that we can get specified output quality.

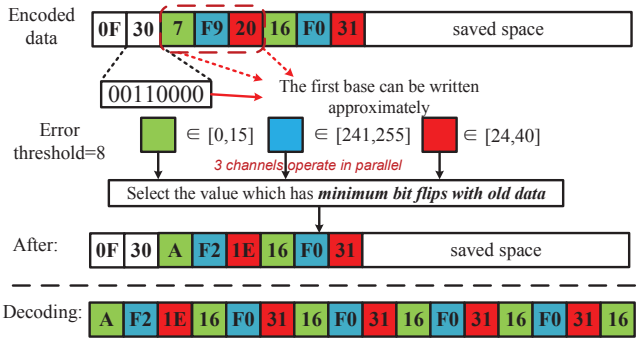

Fig. 6: The example of selecting the energy-efficient base.

Software interface. Following software interface can help programmers set the output image quality. In the function, `start` and `end` are the image data start and end address, respectively. `Q_loss` specifies the required output quality loss. While programmers specify the output quality, APPcache system looks up the quality table to select the corresponding error threshold of the closest quality loss value. Then, with the help of the quality table, our proposed scheme can encode data with this error threshold. The detailed workflow is shown in Fig. 4. Besides, the quality table only includes a few entries, so that the capacity overheads are negligible. A practical way to implement quality control is to extend ISA like [4].

```
set_quality(start,end,Q_loss);
```

D. Overhead Analysis

APPcache needs encoder/decoder to complete the approximate write/read process. We use Synopsys Design Compiler to synthesis the overhead of encoder and decoder with $130nm$ technology node library. The simulation results are scaled down from $130nm$ to $22nm$ according to the scaling rule of transistor. The latency of the encoder and decoder are $1.63ns$ and $0.33ns$, respectively. The encoding latency can be hidden by the long write latency of STT-MRAM, and only the decoding process is on the critical path [18]. However, the performance loss caused by the low decoding latency can be neglected. Besides, the energy parameters of encoder and decoder are $3.84pJ$ and $0.31pJ$, respectively, which are much lower than the write energy of STT-MRAM. The area overhead of encoder and decoder is so small compared with the STT-MRAM cache, which is negligible. Besides, the quality table is accessed only when programmers use the software interface, and the overhead caused by a small number of access operations can be ignored.

IV. Evaluation

We use a full system simulator gem5 [19] to implement our proposed scheme. The system configurations are listed in TABLE I. We use an 8MB STT-MRAM cache as the L2 unified cache. The detailed parameters of STT-MRAM are generated from a circuit-level simulator NVsim [20]. Then, we integrate the timing and energy parameters of STT-MRAM cache into gem5. We evaluate the write energy and raw data compression ratio of several approximate applications (jpeg, sobel, kmeans are from AxBench [21], 2dconv, histeq are from PERFECT [22], and smoothing is from MiBench [23]). A Convolution Neural Network (CNN) based digit recognition is also used in our test benchmarks, and the data set is MNIST [24]. The

details of these benchmarks are listed in TABLE II. To evaluate the output quality, we use Root Mean Square Error (RMSE) [21] as the evaluation metric of *jpeg, kmeans, sobel, 2dconv, histeq, smoothing*, the definition of RMSE is shown in Equation 7, and x_i and y_i are the precise and approximate outputs, respectively. As for digit recognition, we use classification accuracy as the evaluation metric. The quality loss is evaluated via the relative error rate compared with precise output.

$$RMSE = \frac{1}{m}\sum_{i=1}^{m}(x_i - y_i)^2 \quad (7)$$

We compared several aggressive schemes with our proposed APPcache. For the approximate computing applications, several advanced schemes are used to reduce the bit flips of the raw data. As for the non-image raw data, we assign Flip-N-Write [25] encoding with each 8 data bits with 1 tag bit to reduce bit flips. To ensure the fairness of these several schemes, the error rate is set to the same value. The brief introduction of these schemes are as follows:

- **Biscaling** [11]: This scheme uses bidirectional bits truncating to reduce the overall bit flips.
- **AxDedup** [5]: This scheme sets the first element as the only base to reduce the overall data size via data similarity.
- **SimCom** [12]: This scheme sets several flexible bases to eliminate more similar elements.
- **APPcache**: This is our proposed scheme with all optimizations.

TABLE I: System configurations.

Cores	4-Core, 2.0GHz, out-of-order
L1 I/D cache	private, 64KB per core, 2-way; LRU, 2-cycle latency
L2 unified cache	8MB, 64B cache line; 8-way, LRU, 15-cycle latency
Main Memory	4GB, DDR-1600

TABLE II: Application benchmarks.

Application	Algorithm	Dataset	Evaluation Metric
Digit Recognition	cnn	MNIST	Classification Accuracy
Image Compression	jpeg	Axbench	Root Mean Square Error
Edge Detection	sobel		
Image Segmentation	kmeans		
Image Processing	2dconv	PERFECT	
Image Enhancement	histeq		
Image Smoothing	smoothing	MiBench	

A. Write Energy

The normalized write energy is shown in Fig. 7. Due to the efficient encoding, APPcache gains the best write energy reduction. For the Biscaling scheme, the most significant bits cannot always be truncated, thus the energy reduction is limited. Compared with Biscaling, AxDedup and SimCom can reduce similar elements thus reduce write energy, and SimCom gains more energy reduction in all benchmarks except *cnn*. This is because almost all cache lines of *cnn* are full-similar, and in this case, AxDedup can reduce more data size thus leading to more write energy reduction. APPcache can significantly reduce write energy by decreasing more similar elements bit writes. Besides, APPcache can reduce the writes to run_tag and lower the writes for full-similar cache lines. As a result, APPcache can reduce write energy by 31.3%/30.8%/21.9% compared with Biscaling/AxDedup/SimCom with 1% error rate on average. Besides, APPcache can reduce write energy by 36.34%/34.1%/24.03% and 38.05%/33.67%/23.42% compared with Biscaling/AxDedup/SimCom with 3% and 5% error rate, respectively.

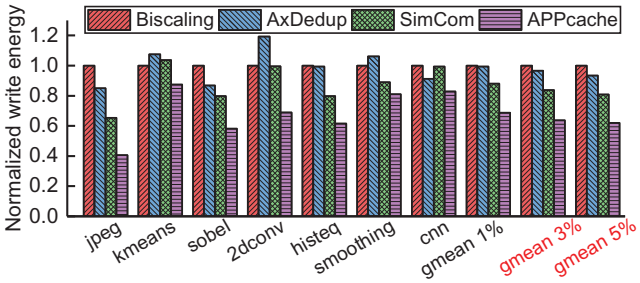

Fig. 7: The normalized write energy of several schemes.

B. Compression Ratio of Image Raw Data

Several schemes can reduce the size of image raw data cache lines. The normalized compression ratio ($\frac{Original\_size}{Compressed\_size}$)of raw data is shown in Fig. 8. Biscsling can reduce the size by truncating bits from both two directions. AxDedup/SimCom can obtain a higher compression ratio by making use of the data similarity. APPcache achieves the largest compression ratio by largely remove similar elements and reduce the tag size of full-similar cache lines. As a result, APPcache can improve the compression ratio of raw data by 296.9%/48.3%/38.0% compared with Biscaling/AxDedup/SimCom with 1% error rate on average. Due to the high compression ratio, our scheme can largely increase the potential cache capacity thus improve the overall system performance [26]. Besides, APPcache can improve the raw data compression ratio by 291.48%/61.38%/39.06% and 275.0%/72.16%/37.55% compared with Biscaling/AxDedup/SimCom with 3% and 5% error rate, respectively.

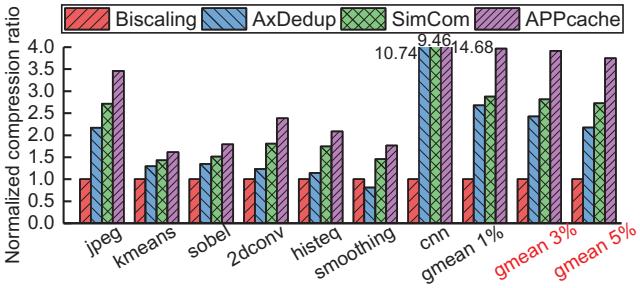

Fig. 8: The compression ratio of the approximate image raw data with several schemes.

C. Output Quality Analysis

In this section, we exhibit the output quality of several benchmarks. Fig. 9 shows the relationship of output quality with different error rates. As we can see, the quality loss increases with a larger error rate, and the quality losses of all benchmarks are within 5% with 1% error rate.

D. Applicability to Different Bitmap Formats and Data Types

In all tested benchmarks, various bitmap formats are used to produce the output result. For *jpeg, kmeans, sobel*, they use RGB bitmap format input file. *histeq, 2dconv, etc.* adopt gray-scale (1 image channel) bitmap image. With the help of

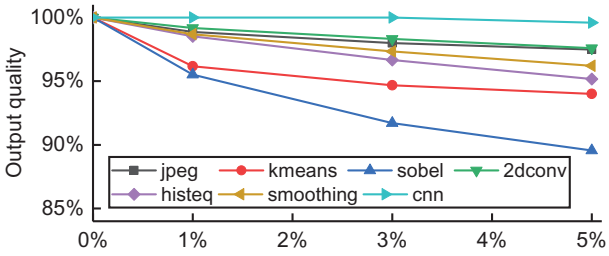

Fig. 9: The output quality with different relative error rate.

annotations, all bitmap formats can be supported in our scheme. As for the image data types, *jpeg* uses uint_16 to store image data. While int data type is used for *kmeans, histeq, etc. cnn, smoothing* use uint_8 to store input data. We assign several bits to store these information in encoded data. In conclusion, our scheme can work well with various bitmap formats and data types in several approximate computing applications.

V. Conclusion

Approximate computing applications lead to large energy consumption and performance demand for the cache memory system. However, traditional SRAM based cache cannot satisfy these demands due to high leakage power and limited density. STT-MRAM is a promising candidate of cache due to low leakage power and good scalability. However, STT-MRAM suffers from high write energy. Thus, we propose APPcache, an STT-MRAM based approximate cache architecture to improve energy efficiency with little output quality loss. Experimental results show that our scheme can reduce write energy by 21.9% and improve the raw data compression ratio by 38.0% compared with the state-of-the-art scheme with the 1% error rate.

Acknowledgment

This work was sponsored by the National Natural Science Foundation of China under Grant 61832007, Grant 61821003, Grant 61772222, and Grant U1705261, in part by the Fundamental Research Funds for the Central Universities No.2019kfyXMBZ037, and National Science and Technology Major Project No. 2017ZX01032-101, and Zhejiang Lab (NO.2020AA3AB07).

References

[1] J. San Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 127–139.

[2] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel, "Approximation-aware multi-level cells stt-ram cache architecture," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. IEEE, 2015, pp. 79–88.

[3] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, 2016.

[4] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "Staxcache: An approximate, energy efficient stt-mram cache," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 356–361.

[5] J. R. Stevens, A. Ranjan, and A. Raghunathan, "Axba: an approximate bus architecture framework," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[6] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[7] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate storage for energy efficient spintronic memories," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[8] A. Jain, P. Hill, S.-C. Lin, M. Khan, M. E. Haque, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[9] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–8.

[10] M. Brandalero, A. C. S. Beck, L. Carro, and M. Shafique, "Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[11] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate memory compression for energy-efficiency," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[12] Z. Chen, Y. Hua, P. Zuo, Y. Sun, and Y. Guo, "Reducing bit writes in non-volatile main memory by similarity-aware compression," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[13] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar, "High-density image storage using approximate memory cells," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 413–426, 2016.

[14] D. Jevdjic, K. Strauss, L. Ceze, and H. S. Malvar, "Approximate storage of compressed and encrypted videos," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 361–373.

[15] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: a cache for approximate computing," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 50–61.

[16] J. San Miguel, J. Albericio, N. E. Jerger, and A. Jaleel, "The bunker cache for spatio-value approximation," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[17] H. Zhao, L. Xue, P. Chi, and J. Zhao, "Approximate image storage with multi-level cell stt-mram main memory," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 268–275.

[18] J. Xu, D. Feng, Y. Hua, W. Tong, J. Liu, C. Li, G. Xu, and Y. Chen, "Adaptive granularity encoding for energy-efficient non-volatile main memory," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[19] 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.

[20] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[21] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, 2016.

[22] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song *et al.*, "Perfect (power efficiency revolution for embedded computing technologies) benchmark suite manual," *Pacific Northwest National Laboratory and Georgia Tech Research Institute*, 2013.

[23] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[24] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[25] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 347–357.

[26] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2012, pp. 377–388.