Improving Performance of TLC RRAM with Compression-Ratio-Aware Data Encoding

Jie Xu, Dan Feng, Yu Hua, Wei Tong*, Jingning Liu, Chunyan Li, and Wen Zhou
Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System
(School of Computer Science and Technology, Huazhong University of Science and Technology)
Ministry of Education of China
*Co-corresponding authors: Wei Tong (Tongwei@hust.edu.cn)
Email:{xujie_dsal, dfeng, csyhua, Tongwei, jnliu, lichunyan, zhouwen}@hust.edu.cn

***Abstract*—Resistive Random Access Memory (RRAM) technology is proposed as a promising replacement candidate for DRAM-based main memory due to its good scalability, low standby power, and non-volatility. The structure of Triple-Level Cell (TLC) can offer higher data density over Single-Level Cell (SLC). However, TLC RRAM suffers from high write energy and latency. Data compression techniques can reduce the size of the data to store. In contrast, data encoding methods such as Incomplete Data Mapping (IDM) can 'expand' the size for latency and energy reduction. We observe that the compression ratio of each cacheline varies, and therefore the saved space of each compressed cacheline is different. On the other hand, we find that different IDMs have different tradeoffs in capacity and write latency/energy. To fully exploit the space saved by compression for reducing the write latency/energy, and improving the performance of TLC RRAM-based main memory system, Compression-Ratio-Aware Data Encoding (CRADE) is proposed. The key idea of CRADE is to dynamically select the best-performing IDM according to the compression ratio of each cacheline. The cacheline is compressed first, and then the compressed cacheline is encoded by IDM. For each compressed cacheline, the IDM which uses the fewest states to encode is applied on the condition that the encoded data size will not exceed the cacheline size. Experimental results show that CRADE can reduce the write energy by 15%, decrease the write latency by 19%, reduce the read latency by 4%, and improve the IPC performance by 2% compared with the state-of-the-art schemes.**

I. Introduction

Non-volatile memory (NVM) technologies such as Phase change memory (PCM), Spin Transfer Torque Random Access Memory (STT-RAM) and Resistive Random Access Memory (RRAM) have emerged as potential replacement candidates of DRAM technology. Among these NVMs, RRAM has the advantages in write latency, energy efficiency and 3D stackable property [1]–[3]. Triple-Level Cell (TLC) RRAM can store three bits of information in a single cell and the density can be further improved than Single-Level Cell (SLC) [4]. Although TLC RRAM can provide outstanding density, its iterative programming method leads to high write energy/latency and results in design challenges for TLC RRAM-based main memory [2]. According to the recent study, the write random-access latency of a 4Mb SLC RRAM testchip is $7.2ns$, while the write random-access latency of Multi-Level Cell is $160ns$ [1]. The write latency of TLC RRAM is higher than both MLC and SLC, and may be more than $160ns$. Besides, the write energy of TLC RRAM is about seven times more than SLC RRAM [2], [5]. The high write energy and write latency are design concerns of TLC RRAM-based main memory.

This work was supported by the National High Technology Research and Development Program (863 Program) No.2015AA015301, NSFC No.61472153, No.61772212, No.61772222, No.61502191. This work was also supported by Engineering Research Center of data storage systems and Technology, Ministry of Education, China.

Data compression techniques, such as frequent pattern compression (FPC) [6] and base-delta-immediate compression [7] can reduce the size of the data to store. In contrast, Incomplete Data Mapping (IDM) [2], which uses several low energy and low latency states out of all the eight states of TLC to encode, can sacrifice the capacity for low latency/energy. In general, q states out of a p-state cell ($q < p$) are used to encode, and r q-state cells are converted into binary digits. This denotes as $IDM((p,q),r)$, as shown in Fig. 3 of Section III-B. If the compression ratio ($size_{original}/size_{compressed}$) of a cacheline is greater than $3/2$, $IDM((8,4),1)$ can be applied to reduce the write energy/latency without incurring memory overhead [5]. We observe that the compressed cacheline sizes are various, and cachelines with higher compression ratios can save more space, while cachelines with lower compression ratios can offer less space. Meanwhile, different IDMs have different tradeoffs in capacity and write latency/energy. The IDM which uses fewer states to encode can sacrifice more capacity for lower latency and energy. To fully exploit the space saved by compression for latency/energy reduction and performance improvement of TLC RRAM-based main memory system, Compression-Ratio-Aware Data Encoding (CRADE) is proposed. CRADE dynamically chooses the best-performing IDM according to the compression ratio of a cacheline. For example, if the compression ratio of a cacheline is greater than 3, $IDM((8,2),1)$ rather than $IDM((8,4),1)$ can be applied to the compressed data for more latency and energy reduction. We have the following contributions in this paper.

- We observe that the compression ratios of cachelines are various, and the IDM which uses fewer states to encode

can sacrifice more capacity for lower latency and energy.

- To maximize the use of the space saved by compression for latency and energy reduction, we propose Compression-Ratio-Aware Data Encoding (CRADE). CRADE dynamically selects the best-performing IDM according to the compression ratio of a cacheline.
- Experimental results show that our scheme can reduce write energy by 15%, decrease the write latency by 19%, reduce the read latency by 4%, and improve the IPC performance by 2% than the state-of-the-art CompEx [5].

The rest of this paper is organized as follows. Section II describes the background of RRAM. Section III presents the observation and motivation. Section IV introduces the design and implementation. Section V and Section VI analyze the experimental results and discuss the related work. Section VII offers conclusions.

II. Background

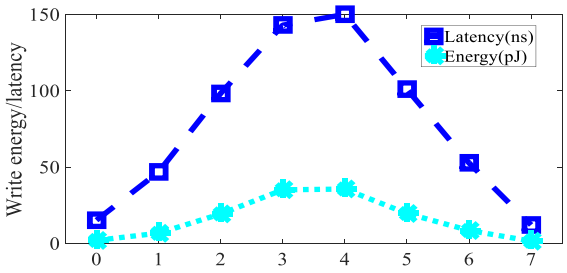
A. The architecture of RRAM

A RRAM cell consists of a top electrode and a bottom electrode, and a metal-oxide layer between them. The logical values are stored in RRAM by changing the resistance of the RRAM cells. The high resistance state (HRS) is used to represent logical value '0', and low resistance state (LRS) is used to represent '1'. In order to change the resistance of a RRAM cell, an external voltage (V_{set} and V_{reset}) is applied across the cell. The switch event from HRS to LRS is called set operation, and the switch event from LRS to HRS is called reset operation. To read the data from a RRAM cell, a small voltage is applied to detect whether the cell is in HRS or LRS.

B. Program-and-verify of TLC

Prior works reported that the resistance differences between HRS and LRS are very large (the ratio of HRS and LRS exceeds 100) [1], [2], and the wide range resistance can be divided into four levels (MLC) or eight levels (TLC) to store two or three bits in a single cell. TLC can offer higher data density than SLC, but its write process is more complex. Program-and-verify (P&V) is commonly used in TLC programming. The P&V programming strategy starts from a set or reset operation, followed by several shorter reset or set pulses. A read operation follows each short write pulse to verify the state of the cell. If the resistance of the RRAM cell reaches the target value, the write operation terminates. Set-to-reset (STR) and Reset-to-set (RTS) are used to reduce the number of iterations. If the Most Significant Bit (MSB) of the target state is '1', we can reach the final states in fewer iterations through reset-to-set [4]. Otherwise, Set-to-reset is used.

The writes to TLC NVMs require much higher energy and latency due to the iterative P&V method. The write energy and latency of TLC are several times more than SLC [2], [5]. The iterative P&V method leads to the characteristic that the write latency and write energy of TLC are dependent on the data written into the cell, as shown in Fig. 1. A TLC has eight resistance states, and the states on the side (e.g., state '0' and '7') of the TLC require fewer iterations, while the states in the middle (e.g., state '3' and '4') need more iterations. For example, programming resistance states '7' and '0' need only a single set and reset operation, while programming resistance states '3' and '4' require several iterations apart from reset and set. More iterations will result in higher write latency/energy, and therefore programming the states on the side needs shorter latency and less energy than the middle states. As illustrated in Fig. 1, the write latency/energy of resistance states '3' and '4' are the highest, while the write latency/energy of '0' and '7' are the lowest.



Cell States	0	1	2	3	4	5	6	7
Latency(ns)	15.2	46.8	98.3	143	150	101	52.7	12.1
Energy(pJ)	2	6.7	19.3	35.1	35.6	19.6	8.5	1.5

Reset-to-set(RTS) Set-to-reset(STR)

Fig. 1. The write latency/energy of different resistance states [2], [5].

III. Motivation

Different cachelines have various compression ratios. For those cachelines with higher compression ratios, more space is saved. Different IDMs have different tradeoffs in capacity and write energy/latency. Compression ratios and IDMs can be delicately combined to maximize the use of the space saved by compression for the most latency and energy reduction.

A. Compression ratio varies

Different compression techniques can be used for bit-write reduction. We evaluate frequent pattern compression (FPC) [6] in this work because of its high performance and low implementation overhead. FPC is proposed for 32-bit words, and can be extended for 64-bit words [5]. Table I lists the data patterns that 64-bit FPC can compress. The compressed words are stored along with their 3-bit prefixes. A 64-bit word can be compressed to 3, 11, 19 or 35 bits. For a cacheline which consists of eight words, each word is compressed separately and the compressed size of each word is also different. The total number of bits of the cacheline may range from 24 to 512. To get the sizes of compressed cachelines, we do experiment and Fig. 2 shows the distribution of compressed cacheline sizes for a set of representative benchmarks. The details of our evaluation are described in Section V-A. The sizes of compressed cachelines vary significantly. Some of them are smaller than a word, while some others are more than seven words. For example, in the cactusADM benchmark, the sizes

of about 25% compressed cachelines are smaller than a word and about 30% are between seven words and eight words. The compression ratio is the ratio of original size and compressed size. The compressed cacheline size varies, and therefore the compression ratio is various. The compression ratio may range from 1 to 64/3 (512/24).

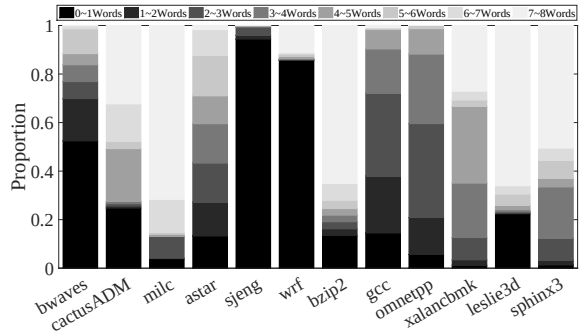


Fig. 2. Compressed cacheline size distribution for 12 benchmarks with the FPC [6] compression algorithm.

B. Tradeoffs exist in different IDMs

Incomplete Data Mapping (IDM) [2] which maps only part of TLC RRAM resistance states into binary values can reduce the write latency/energy of TLC RRAM. As shown in Fig. 3, the q states out of a p-state cell ($q < p$) are used to encode, and r q-state cells are converted into binary digits. This denotes as $IDM((p, q), r)$. Complete Data Mapping (CDM) is the opposite of IDM, which uses all the p states to encode. The similar encoding method 'expansion coding' is also used in CompEx [5].

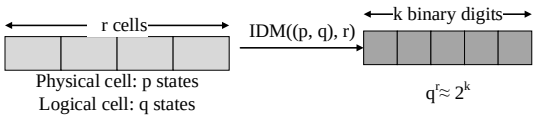


Fig. 3. The coding of $IDM((p, q), r)$. q states out of a p-state cell ($q < p$) are used to encode, and r q-state cells are converted into binary digits.

IDM is different from morphable MLC [8]–[16] which dynamically configured MLC as SLC, Tri-state Cell or MLC, and Fig. 4 illustrates the difference between morphable MLC and $IDM((4, 2), 1)$. Morphable MLC should support the programming strategies of MLC, Tri-state Cell and SLC. To reconfigure MLC to SLC or Tri-state Cell, the set or reset pulse generators to program the cells and the associated iteration control logic should be modified [10], [16]. IDM uses the low energy/latency states ('00' and '11') to encode, and incurs no modification of the read or write circuit.

By eliminating some latency/energy critical states, the write latency and the average write energy are significantly reduced. However, IDM will sacrifice the capacity of TLC RRAM because only a part of the states are used. An example is illustrated in Fig. 5. The data to write are binary '000', '001', '010' ... '111'. Eight CDM (Complete Data Mapping) cells

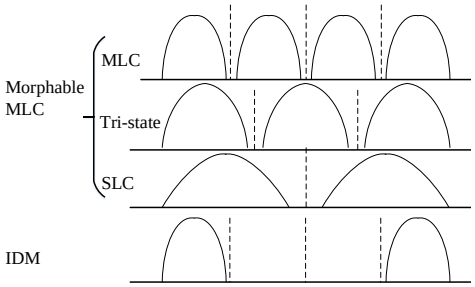


Fig. 4. The resistance states of morphable MLC and $IDM((4, 2), 1)$.

with resistance states S0 (stores '001'), S1 (stores '001') ... S7 (stores '111') can store all the binary digits. If only two low latency/energy states (S0 and S7) are used ($IDM((8, 2), 1)$), each 1-bit binary digit needs an individual IDM cell to store and twenty-four cells are consumed. $IDM((8, 2), 1)$ consumes twice more capacity than CDM, however the write energy and latency are decreased by 67% and 90%, respectively.

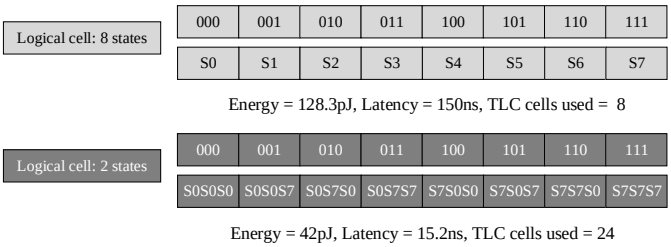


Fig. 5. The energy/latency/capacity comparison of CDM and $IDM((8, 2), 1)$.

The IDM which uses fewer states to encode can sacrifice more capacity for more write latency and energy reduction. Equation 1 is used to evaluate the write energy, write latency and capacity of different IDMs. In Equation 1, $Energy$ and $Latency$ are the latency and energy of the eight states arranged from the smallest to largest. k is the number of states used to encode, and E_k, L_k, C_k are the energy, latency and capacity normalized to CDM TLC cells. Fig. 6 describes the relationship between write energy, write latency, capacity and the number of states used to encode. All the values of energy/latency/capacity are normalized to Complete Data Mapping (CDM) TLC RRAM. As shown in Fig. 6, the capacity, latency and energy increase with the number of states. The write latency and energy are the smallest when two states are used to encode ($IDM((8, 2), 1)$), which are about 90% and 67% smaller than the maximum latency and energy. However, the capacity is also the smallest when two states are used, and is about one third of the maximum capacity.

$$\begin{aligned} E_k &= (\sum_{i=1}^{k} Energy_i)/(k * \log_2 k) \\ L_k &= Latency_k \\ C_k &= log_2(k)/3 \end{aligned} \tag{1}$$

TABLE I
THE DATA PATTERNS 64-BIT FPC CAN COMPRESS [5], [6]. THE 3-BIT PREFIX IS INDICATED IN RED COLOR.

Prefix	Pattern encoded	Example	Compressed	Encoded size
000	Zero run	0x0000000000000000	0x0	3 bits
001	8-bits sign-extended	0x000000000000007F	0x17F	11 bits
010	16-bits sign-extended	0xFFFFFFFFFFFFB6B6	0x2B6B6	19 bits
011	Half-word sign-extended	0x0000000076543210	0x376543210	35 bits
100	Half-word, padded with a zero half-word	0x7654321000000000	0x476543210	35 bits
101	Two half-words, each two bytes sign-extended	0xFFFFBEEF00003CAB	0x5BEEF3CAB	35 bits
110	Consisting of four repeated double bytes	0xCAFECAFECAFECAFE	0x6CAFE	19 bits

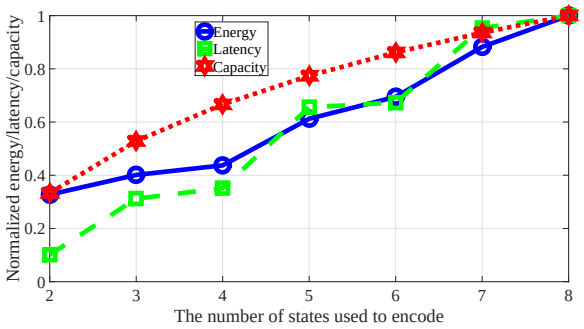


Fig. 6. The tradeoffs of write energy, write latency and capacity in different IDMs.

IV. DESIGN AND IMPLEMENTATION

The compression ratios of cachelines are various. Some cachelines have higher compression ratios, while other cachelines may have lower compression ratios. No matter what's the compression ratio of a cacheline, there is a best-performing IDM which can fit well with the compressed data. For example, $IDM((8,2),1)$ will fit well with those cachelines whose compressed sizes are smaller than two words. Even though $IDM((8,2),1)$ is applied, their encoded sizes will not exceed eight words. On the other hand, $IDM((8,2),1)$ can reduce the more write latency/energy than other IDMs. $IDM((8,2),1)$ cannot be applied to these compressed cachelines between four words and five words, because the size of data after IDM encoding will exceed eight words. Instead, $IDM((8,6),2)$ will fit well with those compressed cachelines between four words and five words. The best-performing IDM should be dynamically selected according to the compression ratios of cachelines. This motivates us to propose Compression-Ratio-Aware Data Encoding (CRADE).

A. The design of CRADE

CRADE selectively applies the best-performing IDM to each compressed cacheline according to the compression ratio of the cacheline. A cacheline consists of eight words, and is compressed word by word. Some words can be compressed, and some cannot. If each word is encoded by IDM separately, the write latency of the compressed word will reduce, while the write latency of uncompressible word will not reduce. However, the write latency of a cacheline is determined by the worst-performance cell of the whole cacheline. Therefore, the write latency of a cacheline is restricted by the slightly compressed word if IDM is applied at the granularity of word. To decrease the write latency of the cacheline, IDM should be used at the granularity of cacheline. After the compression of each word, the compressed sizes of the eight words are added to calculate the compressed cacheline size. The compression ratio can be obtained according to the compressed cacheline size. Then, the IDM which uses the fewest states to encode is applied to the compressed cacheline on the condition that the encoded data size will not exceed the cacheline size. Cachelines with different compression ratios are encoded by different IDMs. For example, if the compression ratio of a cacheline is greater than 3, $IDM((8,2),1)$ will yield the most energy and latency benefits of compression and incur no memory overhead. Table II illustrates the best-performing IDMs used according to the compression ratios. Four different IDMs are evaluated in this paper. The remaining two IDMs which use five states and seven states out of TLC are not adopted, because they have nearly the same improvements as $IDM((8,6),2)$ and CDM, and it's difficult to convert five states and seven states to binary digits.

Additional 8-bit compression tag is required to indicate whether a word is compressed or not. The 8-bit compression tag is considered when calculating the compression ratio, i.e., compression ratio is '(512+8)/(compressed cacheline size+8)'. The 512-bit cacheline is encoded or decoded by IDM together with the compression tag. Another TLC cell (IDM type tag) is needed to indicate which IDM method is used. To ensure that the IDM type tag is not the bottleneck of write operation, the states '7', '0', '1', '6' and '2' are used to represent $IDM((8,2),1)$, $IDM((8,3),2)$, $IDM((8,4),1)$, $IDM((8,6),2)$ and CDM respectively. Fig. 7 describes the percentage of cachelines different IDM methods can be used. About 35% cachelines can use $IDM((8,2),1)$ to encode, and about 12% cachelines cannot be encoded by any IDM.

B. Implementation

The architecture of CRADE is illustrated in Fig. 8. The CRADE logic consists of Encoder and Decoder, and is implemented inside the NVM controller. The Encoder and Decoder are on the write path and read path respectively.

When the TLC RRAM-based main memory receives a write request from the processor, the Encoder works as Fig. 9 illustrates. The cacheline is first sent to the compression

TABLE II
FOUR DIFFERENT IDMS ARE EVALUATED.

Compression ratio	IDM method	Percentage	Energy	Latency
[3, 65/4]	$IDM((8,2),1)$	35%	0.33	0.10
[2, 3)	$IDM((8,3),2)$	20%	0.40	0.31
[3/2, 2)	$IDM((8,4),1)$	20%	0.44	0.35
[6/5, 3/2)	$IDM((8,6),2)$	13%	0.69	0.67
[1, 6/5)	CDM	12%	1	1

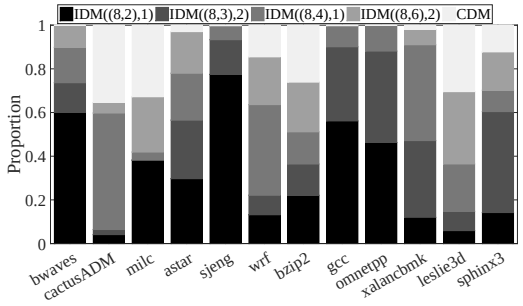


Fig. 7. The proportion of different IDMs can encode in different benchmarks.

logic to attempt data compression. Each word of a cacheline is compared with the data patterns shown in Table I. The corresponding compression tag bit is set if the word is compressible. The eight compressed words are compacted and stored contiguously. Then the total number of bits of the compressed cacheline is calculated. The best-performing IDM method is selected to 'expand' the compressed data according to the compression ratio. The 8-bit compression tag and the 512-bit cacheline are encoded by IDM together. The type of IDM used is stored in an additional cell (IDM type tag). Uncompressible cachelines are sent to the write circuit without compression or encoding.

When the TLC RRAM-based main memory receives a read access from the processor, the Decoder works as Fig. 10 shows. The IDM type tag, compression tag and 512-bit cacheline stored in the TLC RRAM array are read, and the cacheline with 8-bit compression tag is decoded by IDM according to the value of IDM type tag. After that, the compression tag bit and the prefix of each word are used to decompress the cacheline word by word. Uncompressed cachelines are sent to the read buffer without decoding or decompression.

C. Overhead

The circuit of Encoder and Decoder incurs additional space and latency overhead. The Encoder or Decoder consists of three parts, i.e., FPC, Multiplexer and IDM. The Multiplexer calculates the total number of bits of a compressed cacheline and selects the correct IDM. We use Synopsys Design Compiler to evaluate the overhead of the Multiplexer in $130nm$ technology. The Multiplexer incurs 891 gates, $1.75ns$ encoding latency. The circuit overhead of Multiplexer will be

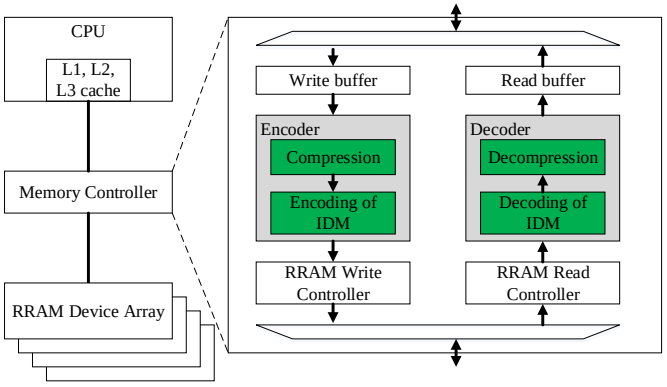


Fig. 8. The architecture of CRADE.

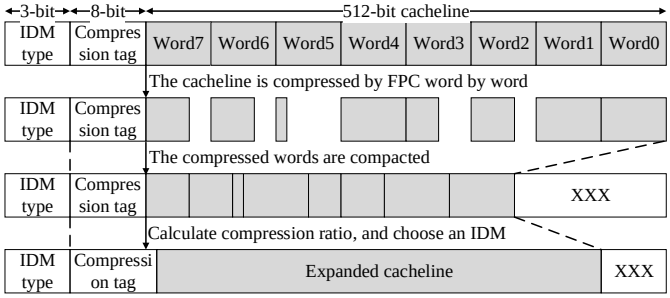


Fig. 9. The Encoder compresses the cacheline and encodes the compressed data using IDM.

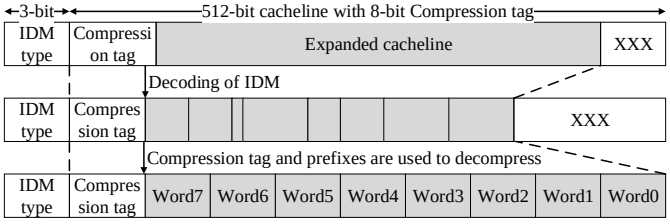


Fig. 10. The Decoder decodes the 'expanded cacheline' and decompresses the compressed cacheline.

smaller when the technology node is scaled down to $22nm$. We estimate the overhead of FPC and IDM based on the prior works [2], [5], [6]. FPC [6] is a simple and widely used compression scheme that has a low compression and decompression overhead. The latencies of compression and decompression are estimated to be $2ns$ and $1ns$ [5], [6]. The encoding and decoding of IDM need another $1ns$ [2]. The latency of Encoder (Decoder) is estimated to be $5ns$ ($2ns$). The Encoder or Decoder will not affect the uncompressible cachelines, because uncompressible cachelines will not pass through the Encoder or Decoder. The compressible cachelines will pass through the Encoder and Decoder. However, the write latency will reduce due to the encoding of IDM. The estimated logic overhead of FPC and $IDM((8,4),1)$ is about $10k$ gates, which is less than 0.1% of the TLC RRAM-based main memory [5]. Our method has three other IDM choices,

and the logic overhead will not exceed 0.4% of the NVM module.

Besides, our scheme incurs small capacity overhead. Each word needs a compression tag bit to indicate whether it's compressed or not, and an additional IDM type cell is used to indicate the IDM type of a cacheline. The overall capacity overhead is 11 bits, and is about 2.1% of a cacheline.

V. Experiment

A. Experiment setup

We evaluate our schemes using a cycle-accurate system simulator Gem5 [17]. The main memory model is based on NVMain [18]. NVMain is a cycle-level main memory simulator designed to simulate emerging NVMs at the architectural level and the RRAM controller is modified to support the Encoder and Decoder. The configuration of the target system is given in Table III. The system is based on a four-core processor. The L1 instruction/data cache and L2 cache are private 32KB, 2 way set-associative and private 1MB, 8 way set-associative respectively. The size of cacheline in L1 and L2 cache is 64 Bytes. The L3 cache is 16 way set-associative and shared by four cores. The size of L3 cache is 16MB. Twelve benchmarks selected from SPEC CPU 2006 [19] are used in our experiment. For each benchmark, one billion instructions are simulated. We evaluate three different schemes in four-core systems respectively:

- FPC + $IDM((8,4),1)$ [5]: $IDM((8,4),1)$ is applied to the compressed data only if the compression ratio is greater than 3/2.
- FPC + $IDM((8,6),2)$: $IDM((8,6),2)$ is applied to the compressed data only if the compression ratio is greater than 6/5.
- CRADE: The best-performing IDM is selected according to the compression ratio of the cacheline.

TABLE III
System configurations

Cores	4-Core, 3.2GHz, out-of-order
L1 I/D cache	private, 32KB SRAM per core LRU, 64B cacheline, 2-way write back, 2-cycle access latency
L2 Cache	private, 1MB SRAM per core LRU, 64B cacheline, 8-way write back, 20-cycle access latency
L3 Cache	16MB SRAM/4-core shared, 64B cacheline, 16-way write back, 50-cycle access latency
Memory Controller	FRFCFS
Memory Organization	4GB TLC RRAM Read 25ns, 4 channels, 4 banks

B. Experimental results

The IPC performance, write latency, write energy and read latency are evaluated in this part. All the values are normalized to the CompEx (FPC + $IDM((8,4),1)$) [5].

1) IPC performance: Fig. 11 shows the normalized IPC for each benchmark. The average IPC improvements of FPC + $IDM((8,6),2)$ and CRADE are -2% and 2%. Comparing FPC + $IDM((8,4),1)$ with FPC + $IDM((8,6),2)$, we can find that a certain IDM may fit well with some benchmarks, but performs badly in other benchmarks. FPC + $IDM((8,6),2)$ outperforms FPC + $IDM((8,4),1)$ in sjeng, wrf, leslie3d and sphinx3, but FPC + $IDM((8,6),2)$ falls below FPC + $IDM((8,4),1)$ in other benchmarks. In leslie3d, the compression ratios of most cachelines (22%) are between 6/5 and 3/2. $IDM((8,4),1)$ cannot apply to these cachelines, while $IDM((8,6),2)$ can fit well with these cachelines. Therefore, FPC + $IDM((8,6),2)$ outperforms FPC + $IDM((8,4),1)$ in leslie3d. CRADE outperforms FPC + $IDM((8,4),1)$ and FPC + $IDM((8,6),2)$ in almost all the benchmarks, because cachelines with different compression ratios can benefit most from the space saved by compression. In bwaves, the IPC improvement is the most significant (9.5%), because the compression ratios of 60% cachelines are greater than 3 and the IPC benefits more from $IDM((8,2),1)$ than $IDM((8,4),1)$. The IPC has almost no improvement in xalancbmk because the compression ratios of most cachelines (44%) are between 3/2 and 2, and $IDM((8,4),1)$ can fit well with these cachelines.

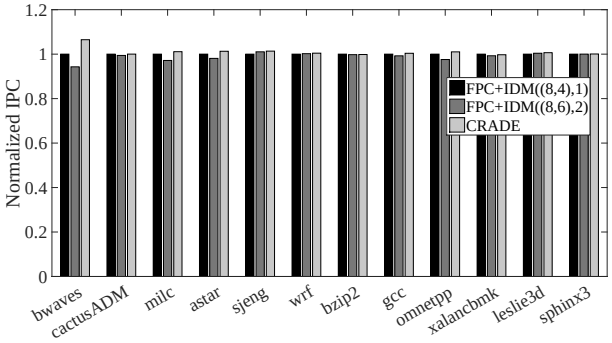


Fig. 11. IPC of different benchmarks (normalized to FPC + $IDM((8,4),1)$).

2) Write latency: The latency of each write access is determined by the slowest cell of a cacheline. IDM can reduce the maximum write latency of the RRAM cells, and therefore the write latency of a cacheline will decrease. Fig. 12 illustrates the write latency of the three different schemes. The average write latency reduction of FPC + $IDM((8,6),2)$ and CRADE is -28% and 19%. FPC + $IDM((8,6),2)$ outperforms FPC + $IDM((8,4),1)$ in leslie3d, but falls below FPC + $IDM((8,4),1)$ in other benchmarks. CRADE outperforms both FPC + $IDM((8,6),2)$ and FPC + $IDM((8,4),1)$ in all the benchmarks, because cachelines with different compression ratios can make full use of the saved space to decrease the write latency. CRADE reduces the write latency by about 50% (the maximum reduction) in bwaves, gcc and omnetpp.

3) Write energy: Fig. 13 describes the write energy of the three different schemes. The average write energy reduction of FPC + $IDM((8,6),2)$ and CRADE is -22% and 15%. FPC + $IDM((8,6),2)$ can reduce more write energy than FPC +

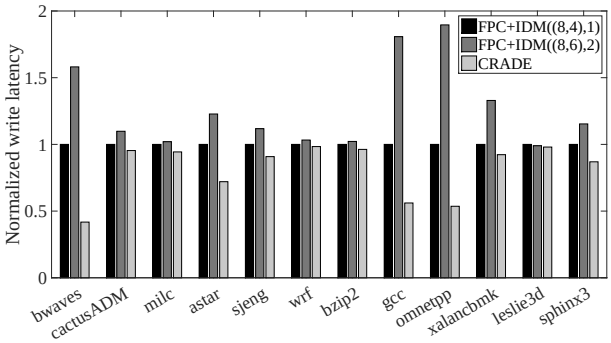


Fig. 12. The write latency of different benchmarks (normalized to FPC + $IDM((8,4),1)$)

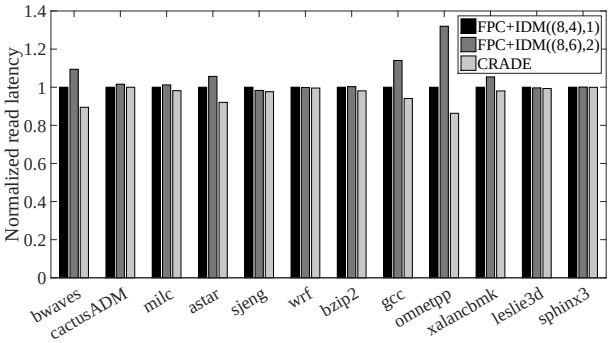


Fig. 14. The read latency of different benchmarks (normalized to FPC + $IDM((8,4),1)$)

$IDM((8,4),1)$ in milc, wrf and leslie3d. The reduction of write energy is similar to write latency. Comparing CRADE with FPC + $IDM((8,4),1)$, the write energy is decreased in all the benchmarks.

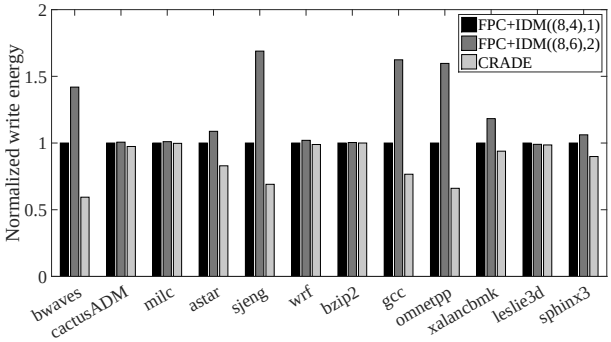


Fig. 13. The write energy consumption of different benchmarks (normalized to FPC + $IDM((8,4),1)$)

4) Read latency: Read operations are on the critical path of the whole system performance, and short read latency will boost the IPC performance. Fig. 14 illustrates the read latency of the three different schemes. The average read latency reduction of FPC + $IDM((8,6),2)$ and CRADE is -6% and 4%. CRADE outperforms both FPC + $IDM((8,6),2)$ and FPC + $IDM((8,4),1)$ in all the benchmarks. In CRADE, the reduction of read latency is the most (about 10%) in bwaves and omnetpp. Our scheme can decrease the write latency, and read requests will benefit from the reduction of waiting time. Therefore, the read latency can be reduced.

VI. Related work

There have been many studies on improving the performance of MLC/TLC NVMs, and the related work can be divided into three categories.

A. State remapping

State remapping was proposed based on the observation that the write energy and latency of MLC/TLC NVMs are significantly dependent on the value written into the cell. For example, writing '10' consumes twenty-five times more energy than '11' in MLC PCM [20]. The most frequently appearing data patterns should correspond to the most energy-efficient resistance states. Static and dynamic mapping methods have been proposed. Some researchers [20]–[22] proposed to map the frequent data patterns to the energy-efficient resistance states at runtime. Some other researchers [23], [24] proposed static remapping techniques with low implementation overhead.

B. Morphable MLC

There are compromises between capacity and energy/latency on designing NVM cells. MLC and TLC offer high capacity, but they suffer from high write energy/latency. MLC can be configured as SLC or Tri-state Cell to guarantee the reliability and the fast access speed when large capacity is not necessary. Dong et al. [16] proposed a circuit-level adaptive MLC/SLC PCM array. Qureshi et al. [11] configured MLC as MLC or SLC to meet the memory requirements of high capacity or low latency. Arjomand et al. [10] coded redundant zero MLC cells into limited bits that were storable in the SLC form. Jiang et al. [13] proposed storing highly compressible cachelines in the SLC form. Nak Hee Seong et al. [12] proposed Tri-state-cell PCM to improve the reliability of PCM. Jiang et al. [14] adopted data compression and proposed fraction encoding to store compressed data using 2, 3 or 4 states MLC. Lee et al. [15] proposed a compression-based hybrid MLC/SLC PCM management technique to obtain the performance of SLC with capacity of MLC simultaneously. Zhang et al. [8] utilized only three states of MLC, and involved only fast state transitions by proactive set. Mengying Zhao et al. [9] proposed SLC-enabled wear leveling scheme through dynamic and adaptive mode transformation from MLC to SLC. Additional circuit and architectural changes are needed to dynamically configure MLC as SLC or Tri-state Cell [5], [10], [16], and the design of NVM cells will become more complex.

C. Incomplete data mapping

Incomplete data mapping (IDM) [2], [5] can reduce the write energy/latency of MLC/TLC NVMs without incurring any modification of TLC RRAM circuit. Liu et al. [2] proposed incomplete data mapping (IDM) which maps only part of

TLC RRAM resistance states into binary values. For a TLC RRAM cell, the 2 most critical states (state '3' and '4') are eliminated in $IDM((8,6),2)$, and two six-state cells can represent 5 ($\log_2(6*6) \approx 5$) digit bits. Two TLCs could have represented 6 digit bits, and therefore $IDM((8,6),2)$ incurs 20% extra space overhead. CompEx [5] is the first work to integrate data compression with 'expansion coding' (the same as IDM) to reduce write energy and latency of TLC RRAM. CompEx is made up of two parts, i.e., compression and expansion coding. Only one type of expansion coding ($IDM((8,4),1)$) is used in CompEx. To ensure the data after expansion coding will not exceed the cacheline size (512 bits), $IDM((8,4),1)$ is applied only if the size of compressed cacheline is smaller than 341 ($2*512/3$) bits. CompEx cannot fully exploit the space saved by compression for latency and energy reduction, because the compression ratios and the saved space are various. To maximum the use of the space saved by compression for energy and latency reduction, CRADE selects the best-performing IDM according to the compression ratio of the cacheline, and CRADE shows better performance than CompEx. Besides, we use FPC in cacheline-level write other than word-level write in CompEx. However, CRADE incurs 2.1% capacity overhead than CompEx.

VII. Conclusion

In this paper, we propose Compression-Ratio-Aware Data Encoding (CRADE) to reduce the write latency/energy and improve the IPC performance of TLC RRAM-based main memory system. CRADE dynamically chooses the best-performing IDM method according to the compression ratio of the cacheline, and can fully exploit the space saved by compression for latency and energy reduction. CRADE is agnostic to the choice of compression technique, and FPC is applied to compress the cacheline data in our work. Experimental results show that CRADE can reduce the write energy by 15%, decrease the write latency by 19%, reduce the read latency by 4%, and improve the IPC performance by 2% with only 2.1% capacity overhead.

References

[1] S.-S. Sheu *et al.*, "A 4mb embedded slc resistive-ram macro with 7.2ns read-write random-access time and 160ns mlc-access capability," in *2011 IEEE International Solid-State Circuits Conference*, Feb 2011, pp. 200–202.

[2] D. Niu *et al.*, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 131–137.

[3] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 476–488.

[4] ——, "Understanding the trade-offs in multi-level cell reram memory design," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–6.

[5] P. M. Palangappa and K. Mohanram, "Compex: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvm," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 90–101.

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

[7] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 377–388.

[8] X. Zhang *et al.*, "Tristate-set: Proactive set for improved performance of mlc phase change memories," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 659–665.

[9] M. Zhao *et al.*, "Slc-enabled wear leveling for mlc pcm considering process variation," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[10] M. Arjomand *et al.*, "A morphable phase change memory architecture considering frequent zero values," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. IEEE, 2011, pp. 373–380.

[11] M. K. Qureshi *et al.*, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 153–162.

[12] N. H. Seong *et al.*, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2013, pp. 440–451.

[13] L. Jiang *et al.*, "Improving write operations in mlc phase change memory," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–10.

[14] ——, "Er: Elastic reset for low power and long endurance mlc based phase change memory," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 39–44.

[15] H. G. Lee *et al.*, "A compression-based hybrid mlc/slc management technique for phase-change memory systems," in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*. IEEE, 2012, pp. 386–391.

[16] X. Dong and Y. Xie, "Adams: Adaptive mlc/slc phase-change memory design for file storage," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, Jan 2011, pp. 31–36.

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

[18] M. Poremba *et al.*, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[19] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[20] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 175–182.

[21] P. Chi, C. Xu, X. Zhu, and Y. Xie, "Building energy-efficient multi-level cell stt-mram based cache through dynamic data-resistance encoding," in *Fifteenth International Symposium on Quality Electronic Design*, March 2014, pp. 639–644.

[22] X. Zang *et al.*, "Energy optimization for multi-level cell stt-mram using state remapping," in *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*. IEEE, 2016, pp. 546–553.

[23] M. Zhao *et al.*, "State asymmetry driven state remapping in phase change memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 27–40, 2017.

[24] Y. Chen *et al.*, "Processor caches with multi-level spin-transfer torque ram cells," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*. IEEE Press, 2011, pp. 73–78.