zeroTT: A Two-Step State Transition Avoidance Scheme for MLC STT-RAM

[1]Dong Yin, [1]Huizhang Luo*, [2]Jeff Zhang, [1]Mingxing Duan, [1]Wangdong Yang, [1]Zhuo Tang, [1]Kenli Li*
[1]Hunan University, Changsha, China
[2]Arizona State University, Tempe, United States

ABSTRACT

Compared with conventional SRAM, Spin-Transfer Torque Random Access Memory(STT-RAM) is expected to play a crucial role in future memory technologies with the increasing demands for higher storage density and lower power consumption for modern embedded systems. Moreover, Multi-Level Cell (MLC) STT-RAM outperforms Single-Level Cell (SLC) STT-RAM since it has higher bit density. However, MLC STT-RAM suffers from write performance due to the two-step state transitions (TTs) in memory cells' soft domain. State-of-the-art approaches mitigate this issue by reducing TTs with efficient data coding. Unfortunately, none of the existing works can fully eliminate the TTs. In this work, zeroTT, an optimal $(3,4)$-based expansion coding method that eliminates TTs for MLC STT-RAM. The design of ZeroTT considers space overhead and coding complexity, and our experimental results demonstrate that zeroTT can completely avoid TTs, leading to a more efficient MLC STT-RAM memory in terms of access latency, energy consumption, and device lifetime.

KEYWORDS

MLC STT-RAM, expansion coding, two-step state transitions

1 INTRODUCTION

SRAMs have been used as a conventional cache, providing a hierarchy of memory between the CPU and the main memory for many years. However, SRAM technologies are experiencing scalability and high leakage power consumption issues in advanced technology nodes. Spin Transfer Torque Random Access Memory (STT-RAM) [1] [2], as a form of non-volatile memory that leverages electron spin to store and retrieve data, has the potential to replace SRAM in various applications due to its advantages of high density and lower power leakage. In general, STT-RAM can be classified into two categories: Single Level Cell (SLC) STT-RAM and Multi-Level Cell (MLC) STT-RAM. SLC STT-RAM cell utilizes two resistance states of a magnetic tunneling junction (MTJ) device to store one bit of information. Compared with SLC STT-RAM, MLC STT-RAM achieves higher storage density because it can store multiple bits of information per cell [3]. However, the higher density comes with the write performance cost as the memory cell suffers from two-step state transitions (TTs) where soft and hard domains cannot be changed to two opposite directions simultaneously. TTs significantly affect the access latency, energy consumption, and the lifetime of MLC STT-RAM devices.

Multiple research works have been looking to reduce TTs and improve the performance of MLC STT-RAM. Hsieh *et al.*[4] proposed the TSE scheme to minimize TTs by converting them into soft transitions (STs) or zero transitions (ZTs). They also proposed the AES [5] scheme by using alternative encoding to minimize the occurrence of TTs. Ahmad et al.[6] proposed a scheme to reduce the write energy by reducing the number of TTs and hard transitions (HTs). Xu et al.[7] proposed an ES scheme, which separately encodes the bits of the hard and soft domain to optimize the write energy. Zang *et al.*[8] proposed an encoding scheme that can minimize the occurrence of left bit flips and reduce states '01' and '10' from writing operations. Luo *et al.* [3] proposed the TSTM scheme that improves the STT-RAM device lifetime. These existing coding techniques can significantly reduce TTs, however, to the best of our knowledge, none of them can completely avoid TTs. This paper emphasizes the importance of avoiding TTs in the cache design. A cache line consists of multiple bytes (e.g., 64 bytes for most architectures), where they are accessed in parallel in the granularity of a write unit (i.e., 64 bits for modern machine word length). In the worst case, a TT occurrence will result in a 2X increase in write time during a write unit.

Table 1: Comparison of zeroTT against existing works. zeroTT has a lower space overhead and TT ratio.

	C-MLC [9]	TSTM [3]	AES [5]	zeroTT
(*M*,*N*)	(2,2)	(2,3)	(2,3)	(3,4)
Space overhead	**0**	50.0%	50.0%	33.3%
Coding complexity	**128**	3072	3072	256
TT ratio	0.25	0.0208	0.0208	**0**

To address the write performance issue in MLC STT-RAM devices, we propose zeroTT, a novel encoding method for MLC STT-RAM caches. Our work starts by analyzing and understanding why state-of-the-art coding methods that leverages $(2,3)$-based expansion coding (such as TSTM [3] and AES [5]) can not fully eliminate TTs. In our analysis, we find that in fact the best $(2,3)$-based coding method can minimally reduce the TT ratio (as defined in Seciton 2.3) to 2.08%. To look for an encoding scheme that can fully avoid TTs during memory writes, we further generalize the expansion coding method as (M,N), where M and N are the lengths of the data and code, respectively. Then we proposed zeroTT, an optimal $(3,4)$-based coding method that makes trade-offs between storage overhead and coding complexity to fully eliminate TTs, as shown in Table 1. Lastly, we evaluate zeroTT with 10 benchmarks

*Huizhang Luo is the corresponding author (luohuizhang@hnu.edu.cn).

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3658240

from SPEC CPU Benchmark Suites [10]. Our experimental results demonstrate the effectiveness of zeroTT in terms of access latency, energy consumption, and device lifetime.

The main contributions of this paper are outlined as follows:

- We extensively analyze the existing $(2, 3)$-based coding methods and show that they cannot avoid TTs.
- We propose an optimal zeroTT [1] coding method that can fully eliminate TTs while considering the tradeoffs between the space overhead and the coding complexity.
- Our detailed evaluation shows that zeroTT based on STT-RAM cache design is more efficient in terms of memory access latency and memory energy consumption. We also show that zeroTT can help improve the lifetime of the STT-RAM memory devices.

2 BACKGROUND AND MOTIVATION

This section introduces the background and motivation of the work.

2.1 SLC STT-RAM and MLC STT-RAM

Different from traditional SRAM technologies using electric charge or current flows to store data, STT-RAM uses magnetic storage elements called magnetic tunnel junctions (MTJs), as shown in Fig. 1. An MTJ consists of two ferromagnetic layers separated by an oxide barrier layer, such as MgO [9]. In STT-RAM, the MTJ structure consists of two layers: a reference or fixed layer, which is a permanent magnet set to a specific polarity, and a free layer that can be altered to align with an external magnetic field for data storage. In SLC-RAM MTJ, the resistance is determined by the magnetization direction of the free layer. When the magnetization direction of the free layer is parallel to the reference layer, the MTJ stays at low resistance, denoted by R0. Conversely, if the magnetization direction is not parallel, the MTJ stays at high resistance, denoted by R1, as shown in Fig. 1(a).

The MLC STT-RAM improves upon SLC STT-RAM in terms of bit density. Fig. 1(b) shows a four-level (i.e., 2-bit) MTJ device [11]. In an MLC STT-RAM cell, the free layer is further divided into *soft* and *hard* magnetic domains with distinct magnetic properties. The magnetization direction of the soft domain can be controlled by a small amount of current, whereas a larger current is required for the hard domain. Depending on the possible combinations of the relative magnetization directions of the soft and hard domains in the free layer, there are four possible resistance states, namely, R00, R01, R10, and R11 (as shown in Fig. 1(b)).

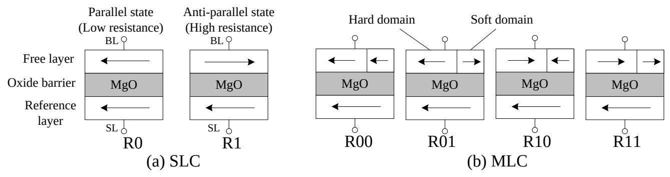


Figure 1: Different MTJ structures: (a) SLC MTJ [12]. (b) Parallelized MLC MTJ [11].

[1]The main part of the source code is available at: https://github.com/yindong2050/STT-RAM

2.2 State Transitions in MLC STT-RAM

MLC STT-RAM offers more density and energy efficiency than the SLC STT-RAM. However, there are more states in each MLC STT-RAM cell. Due to different voltage/current requirements for switching stats in soft and hard domains, The transitions between different MLC STT-RAM cell states exhibit **variable** latencies and energy consumption.

As shown in Fig. 2, the MTJ resistance state transition for a 2-bit (4-level) MLC STT-RAM cell can be classified into four types: Zero Transition (ZT), Soft Transition (ST), Hard Transition (HT), and Two-step State Transition (TT).

- Zero Transition (ZT): MTJ of the STT-RAM cell stays at the original state. No state transition in either the soft domain or hard domain. For example, R00 -> R00;
- Soft Transition (ST): A *LOW* current is applied in the free layer of MTJ, which changes only the magnetic direction of the soft domain while the hard domain stays at its original state. For example, R00 -> R01;
- Hard Transition (HT): A *HIGH* current is applied in the free layer of MTJ, which changes the magnetic direction of both the soft and hard domains. For example, R00 -> R11;
- Two-step state Transition (TT): A combination of a HT and ST transitions to change the magnetic direction of the hard domain. For example, R00 -> R10;

Compared to other state transitions in MLC STT-RAM, TTs incur longer write latency and more energy consumption. For instance, recent works report that the latencies of ZT, ST, HT, and TT are 0 *ns*, 10 *ns*, 10 *ns*, and 20 *ns* [13] respectively; and the write energy of ZT, ST, HT, and TT are 0 nJ, 0.843 nJ, 1.659 nJ, and 2.502 nJ, respectively [14].

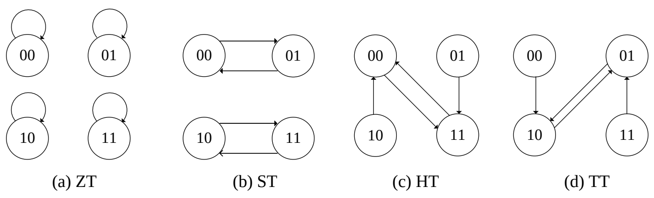


Figure 2: Four types of state transitions in MLC STT-RAM.

2.3 Prior Works and Our Motivation

To deal with the costly TTs in MLC STT-RAMs, Luo et al. proposed TSTM [3] and Jen-Wei Hsieh et al. proposed AES scheme [5] to reduce the occurrence of TTs by efficient data encoding/decoding.

As shown in Fig. 3 (a) and Fig. 3 (b), both TSTM and AES schemes encode 2-bit data into 3-bit codes, which we refer to as **the (2,3)-based coding method**. The (2,3)-based coding method allows that a *single* data item maps to *multiple* codes, such that the actual encoding enables data-dependent minimization of the TTs.

However, we observe that both the (2,3)-based coding methods from TSTM [3] and AES[5] **cannot fully avoid TTs**. Fig. 4(a) shows that, in the TSTM scheme, the original data "10" and "01" are encoded to "101010" and stored in three 2-bit MLC STT-RAM cells. If the data to be written is "00" and "11" (encoded as "000111"), it

Data	Code
00	000
01	001
	010
	100
10	011
	101
	110
11	111

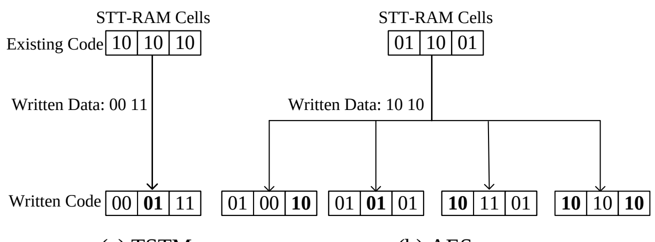
(a) TSTM

Data	Code
00	000
	111
01	001
	110
10	010
	101
11	011
	100

(b) AES

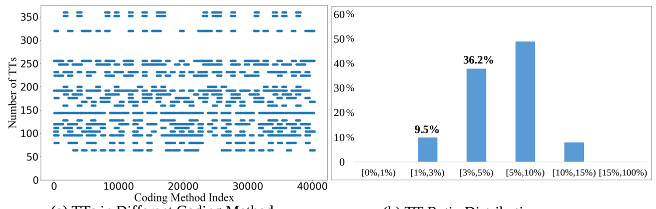
Figure 3: Coding methods of TSTM [3] and AES [5].

will incur the TT in the 2nd MLC STT-RAM cell. Similarly, in the AES scheme, although each 2-bit data item has two options during the encoding, we found that there is no possible way to write data "10" and "10" to existing MLC STT-RAM cells (The initial states for the three cells are 01, 10, 01 respectively.) without incurring TTs. Fig. 4(b) highlights which STT-RAM cell(s) will have TTs in their state transitions.



(a) TSTM (b) AES

Figure 4: Example of TTs in TSTM and AES schemes, highlighting MLC STT-RAM cells with TTs.



(a) TTs in Different Coding Method, The Coding Complexity is 3072

(b) TT Ratio Distribution

Figure 5: Number of TTs and the distribution of TTs across all the possible (2,3)-based coding methods.

Based on this observation, we further ask: *Does there exist a (2,3)-based coding scheme that can fully avoid the TTs?* To answer this question, we did an exhaustive search over a total of $2^3!$=40,320 (2,3)-based coding methods (from AES). For each coding method, we tried to minimize the TTs over all possible pair of existing code (for 2 data items) and the written data code. Fig. 5(a) shows the minimal number of the TTs which are still required for each possible (2,3)-based coding. Note that each (2,3) coding scheme requires 3072 total write tests, as explained in Section 3.4. Fig. 5(b) shows the frequency distribution of the TT ratios (defined as the frequency of TTs/total number of TTs) of all the possible coding methods. We can find that (2,3)-based coding methods can reduce TTs in MLC-RAM cells. In particular, 45.7% of them can reduce the TTs from 25% of baseline C-MLC STT-RAM writes [9] (see Table 1) to less than 5%. Fig. 5 also shows that none of the (2,3)-based coding methods can fully eliminate TTs. The minimum TT ratio is 2.08%, which is achieved by TSTM and AES methods. This observation motivates the design of our zeroTT coding scheme.

3 ZEROTT

In this section, we first generalize the (2,3)-based expansion coding method. Then, we present two approaches to find the zeroTT coding method for different (M, N). Lastly, we propose an optimal $(3, 4)$-based coding method along with an example illustration.

3.1 A General Expansion Coding

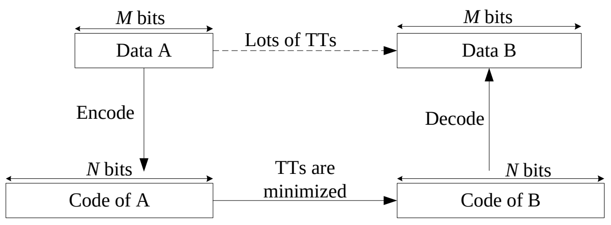


Figure 6: Illustration of (M, N) expansion coding. An M-bit data is encoded to an N-bit code ($M < N$). Expansion coding applies the optimal code to minimize/eliminate TTs.

Expansion Coding: A (M, N) expansion coding, in which $M < N < 2M$, is a coding that encodes data of length M into the codes of length N, such that TTs in writing the codes are less than those in writing the data directly.

We elaborate on the (M, N) expansion coding with the help of Fig. 6. Assuming cache lines are divided into multiple M-bit segments, for each segment, the total number of possible M-bit data words is 2^M. The expansion coding expands the M-bit data word to an N-bit code word. Thus, the total number of possible N-bit codes is 2^N. Note that the total number of codes is larger than that of data, expansion coding can provide multiple codes for each data word. As shown in Fig. 6, given that data A is written to data B, there can be lots of TTs through direct overwriting. Instead, if expansion coding is used to write the data, we can search for the best codes to write the data. To choose the codes with the minimum number of TTs, expansion coding first reads the existing code of A. Next among all available codes of data B, it selects the one that incurs the minimum number of TTs for overwriting the existing code. With this approach, the number of TTs can be reduced significantly. Note that the data density of MLC STT-RAM is partially offset by expansion coding based on MLC STT-RAM since we use N-bit code to present M-bit data ($M < N$). However, the density of expansion coding based on MLC STT-RAM is still higher than SLC STT-RAM ($N < 2M$).

3.2 How to Find zeroTT Coding Methods?

ZeroTT coding methods for different (M, N): Assume the data set is S, and the code set is C. The total numbers of entries in S and C are 2^M, and 2^N, respectively. The coding method of f is defined as $f : S \to C$. Note that one data may have multiple codes. A cluster of codes that is mapped to the same data s is defined as $f_{S \to C}(s)$. We use $(3, 4)$ as an example to explain the details. The data set is $S = \{000, 001, \dots, 111\}$, and the code set is $C = \{0000, 0001, \dots, 1111\}$. The most obvious way is to regard each code and its inverse as a cluster. Thus, the coding method is $f : S \to C = \{000 \to \{0000, 1111\}, 001 \to \{0001, 1110\}, \dots, 111 \to \{0111, 1000\}\}$. The cluster of data 000 is $f_{S \to C}(000) = \{0000, 1111\}$. Let $num(c_i, c_j)$ be the number of TTs to write code c_i into code c_j. The total number of TTs for all possible transitions with a given coding method $f : S \to C$ is defined as:

$$Cost(C) = \sum_{c \in C} \sum_{s \in S} \min\{num(c, f_{S \to C}(s))\} \tag{1}$$

Our goal is to find a coding method that **minimizes the Equation** (1), *among the total number of* $(2^N)!$ *coding methods*. For each (M, N), the role of the coding method is to group all the entries of C into different clusters, each cluster with 2^{N-M} entries. Algorithm 1 traverses all the coding methods and records those that have the minimum value of TTs. The algorithm is summarized as the following: Calculating the value of *Cost* for each iteration P based on Equation (1) (Line 2). If *Cost* is smaller than *Min*, setting the new minimum value, and recording the coding method (Lines 3-5). Then, the algorithm generates the next coding method (Lines 6-13). These steps repeat as a loop until all the coding methods are traversed.

Algorithm 1 Finding the zeroTT coding methods.

```
Require: $P = P_0 P_1, \dots, P_{2^N-1}$ -The initial coding method
    in ascending order, where $P_j = C_j$ and $\{S_0 \to \{P_0, \dots, P_{2^{N-M}-1}\}, \dots, S_{2^M-1} \to \{P_{2^N-2^{N-M}}, \dots, P_{2^N-1}\}\}$ .
    $Min$-The initial minimum cost of $P$.
Ensure: $Q = Q_0 Q_1, \dots, Q_{2^N-1}$ -The best-fit coding method, the
    expressions of $Q$ and $P$ are the same.
 1: while True do
 2:     Calculate the value $Cost$ of $P$ based on Equation (1).
 3:     if $Cost < Min$ then
 4:         Record the solution $Q = P$ and $Min = Cost$.
 5:     end if
 6:     Find the largest index $k$ such that $P_k < P_{k+1}$ in $P$.
 7:     if $k$ does not exist then
 8:         return Coding method $Q$.
 9:     end if
10:     Find the largest index $l$ greater than $k$ such that $P_k < P_l$.
11:     Swap $P_k$ and $P_l$.
12:     Reverse the sequence from $P_{k+1}$ up to the final element
        $P_{2^N-1}$.
13:     $P = P_0 P_1, \dots, P_{k-1} P_l P_{2^N-1}, \dots, P_k, \dots, P_{k+1}$.
14: end while
```

The complexity of calculating Equation (1) is $O(2^{M+2N})$. The complexity of searching the coding method is $O((2^N)!)$. Thus, the overall complexity is $O(2^{M+2N}(2^N)!)$.

3.3 An Efficient Greedy Algorithm

To efficiently find the zeroTT coding method, we propose a greedy algorithm. The main observation of the greedy algorithm is that memory cells do not need to change their values if the original states and target states are in the same cluster. We use this wisdom to reduce the total number of TTs. Our greedy algorithm is described as follows:

1) Choose 2^{N-M} codes from C as the first cluster G_1 mapping to the first element of S, such that $num(G_1) = \sum_{c_i \in G_1} \sum_{c_j \in G_1} num(c_i, c_j)$ has the maximum value.
2) Choose 2^{N-M} codes from the remain of C as the second cluster G_2 mapping to the second element of S, such that $num(G_2) = \sum_{c_i \in G_2} \sum_{c_j \in G_2} num(c_i, c_j)$ has the maximum value.
3) Repeat the steps until all codes are allocated into clusters.

Note that our greedy algorithm avoids the largest numbers of TTs in each cluster, so we have the minimum total number of TTs from Equation(1). The total number of TTs is presented as: $Cost(C) = \sum_{c_i \in C} \sum_{c_j \in C} num(c_i, c_j) - \sum_{i=1}^{2^M} num(G_i)$.

Given M and N, the runtime complexity of our greedy algorithm is $O(2^{M+2N})$.

3.4 Optimal M and N

Another question we ask is: what is the optimal (M, N) expansion coding method that can achieve zeroTT's goal?

Table 2: Comparison of (M, N) expansion coding methods.

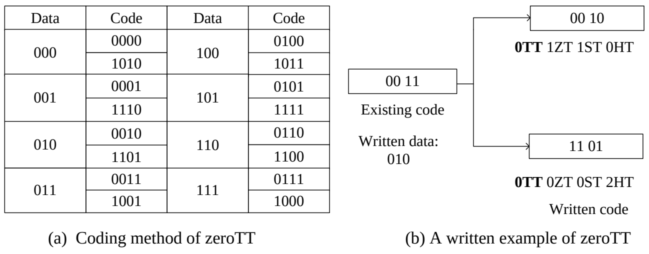
(*M*,*N*)	**(2,3)**	**(3,4)**	**(4,5)**	**(5,6)**	**(6,7)**	**(7,8)**
Space overhead	50.0%	33.3%	25.0%	20.0%	16.7%	14.3%
Coding complexity	3,072	256	1,210,720	6,144	469,762,048	131,072

We find that there is a tradeoff between the storage overhead and the coding complexity in different (M, N) expansion coding methods. The space overhead is calculated as $(N - M)/M$. The coding complexity indicates the complexity of encoding data and decoding code, which are implemented by hardware. Table 2 presents a comparison on different settings of (M, N). To limit the space overhead, we specifically evaluate the scenarios where N equals $M + 1$. Table 2 shows that space overhead decreases slowly as N increases. the coding complexity in Table 2 is calculated as $2^{M+N}(N/2)$. Note that when N is odd, we employ a $2N$-bit code for encoding two data words, and as a result, (M, N) becomes $(2M, 2N)$. This explains why the coding complexity of (2,3) is higher than that of (3,4).

According to Table 2, we choose (3,4) as the final coding scheme by considering the trade-off between space overhead and the coding complexity.

3.5 A Walk-through Example of zeroTT

Fig. 7(a) illustrates an optimal (3,4) zeroTT coding we find from the previous discussion. zeroTT first reads the existing data code and chooses the written codes that do not incur TTs. If more than one zeroTT written code is found, zeroTT selects the one with the lowest energy consumption. As shown in Fig. 7(b), the existing code is "00 11". For the written data "010", we have two candidate written codes “00 10" and “11 01". We can find that both codes do not incur TTs, but the energy consumption for writing code “00 10" is lower [14]. Thus, zeroTT will encode “00 10" for data “010".



Data	Code	Data	Code
000	0000	100	0100
	1010		1011
001	0001	101	0101
	1110		1111
010	0010	110	0110
	1101		1100
011	0011	111	0111
	1001		1000

Figure 7: Illustration of zeroTT.

4 EVALUATION

Experimental Setup. Our experiments run on the MLC STT-RAM based cache simulator [5, 11]. We present a total of 10 representative benchmarks, from SPEC CPU Benchmark Suites [10]. These benchmarks cover code compilation, data compression, partial differential equation solver, mail detection, etc. We simulate the execution of these benchmarks [15] on a two-level cache hierarchy processor in gem5 and collect the memory access traces to feed into STT-RAM cache simulator. Table 3 summarizes the detailed setting.
Baselines. We compare our zeroTT with conventional MLC STT-RAM (C-MLC [9]), TSTM [3], and AES[5].

Table 3: System configuration.

Evaluation Platform	Intel i9-12900H, 14 cores, 2.5GHz
Operating System	Ubuntu 20.04.4 LTS, X86
L1 I/D cache	64KB, 1-way, 64 byte/line, 2-cycle hit LRU replacement policy
L2 data cache	2MB, 4-way, 64 byte/line, 20-cycle hit write back, LRU replacement policy
Main memory	8GB LPDDR5 DRAM, 16 banks 32768 rows/bank, 1024 columns/row, 200-cycle latency
Simulation Software	Gem5 20.1.0; MLC STT-RAM cache simulator
Workloads	10 benchmarks from SPEC CPU 2006 and SPEC CPU 2017

4.1 TTs Evaluation

Fig. 8 compares the number of TTs required by the four methods. Both TSTM and AES can greatly reduce the number of TTs over C-MLC STT-RAM by using expansion coding. However, the average amount of TTs across all benchmarks is still more than 10^7, which will have a significant impact on the MLC STT-RAM write performance. ZeroTT, on the other hand, fundamentally avoids all TTs regardless of the workloads.

4.2 Access Latency Evaluation

For the total access latency evaluation, we follow [13] and set the cache line write delay to 20ns when the memory write incurs a TT in the STT-RAM cell, while the write delay is set to 10ns if it only requires STs or HTs during writing. Through our end-to-end benchmarking, we find from Fig. 9 that zeroTT outperforms C-MLC STT-RAM by an average of 40.1% in latency reduction. When

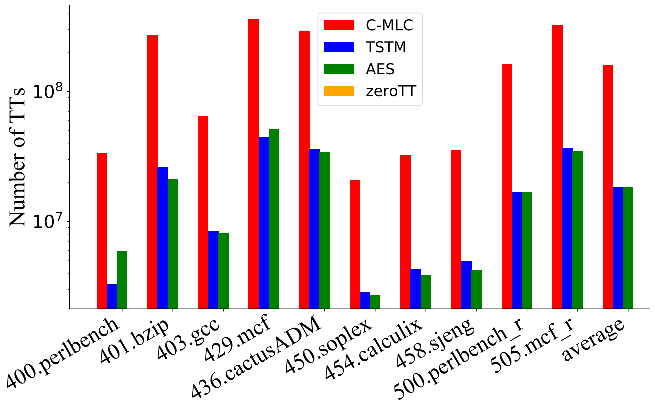


Figure 8: TTs evaluation under four encoding methods.

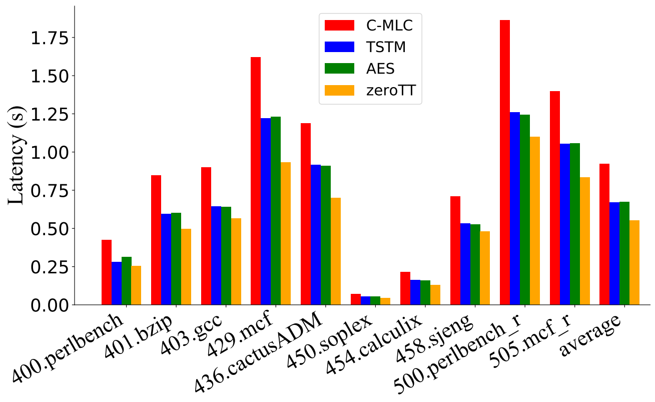


Figure 9: Latency evaluation under the four methods.

Compared with TSTM and AES, zeroTT also shows an average performance improvement of 17.6% and 17.8%, respectively. ZeroTT achieves faster program execution due to its avoidance of TTs in STT-RAM cache writes.

4.3 Energy Consumption Evaluation

Fig. 10 summarizes the total STT-RAM based cache energy consumption with four different coding schemes. In Fig. 10, TSTM experiences higher energy consumption than C-MLC method, which means that the energy overheads from expansion encoding outweigh the energy savings from TT reduction for TSTM method (Although the latency of TSTM is better in Fig. 9). Both AES and zeroTT are more energy-efficient than C-MLC and TSTM due to the greater TT reduction and their energy-aware selection of written code in the designs. When compared with AES, zeroTT consumes less energy for the majority of the benchmarks and is the most energy-efficient design on average. In certain cases, zeroTT costs more energy, e.g., 500.*perlbench_r* and 458.*sjeng*. The reason is that the actual memory energy consumption of a particular program highly depends on the data patterns and the encoding scheme. For example, considering the write of “000000 000101" as “000010 010110": assume the existing code in STT-RAM is “000000000 000001001", AES encodes the written data as “000000010 001001101". During

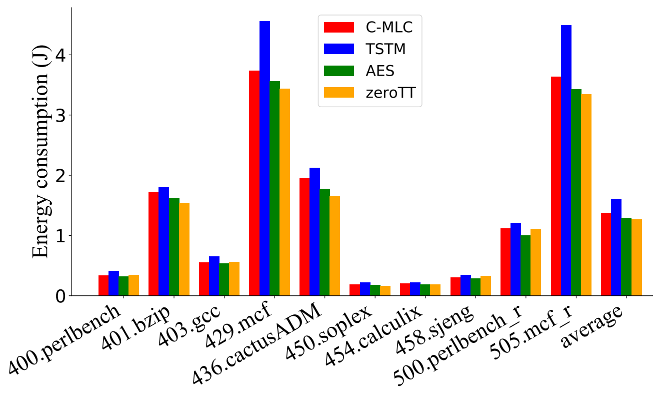


Figure 10: Energy consumption under four encoding methods.

the write, it leads to 6 ZTs and 3 STs in the STT-RAM cells, so this write energy consumption is 2.529nJ [14]. However, for zeroTT, the same existing data are encoded as "00000000 00000101", and the same written data are encoded as "00001101 11011100". This zeroTT write has 2 ZTs, 3 STs, and 3 HTs, incurring an energy consumption of 7.506nJ. While in another case, where we are writing "010111 000110" as "110011 101001", zeroTT consumes 5.031nJ with 3 ZTs, 4 STs, and 1 HT, outperforming AES which costs 8.376nJ with 2 ZTs, 5 STs, 1 HT, and 1 TT. Nevertheless, zeroTT reduces memory energy consumption by an average of 7.7%, 20.8% and 1.6%, when compared with C-MLC STT-RAM, TSTM and AES respectively.

4.4 Lifetime Evaluation: Total Number of Cell Flips

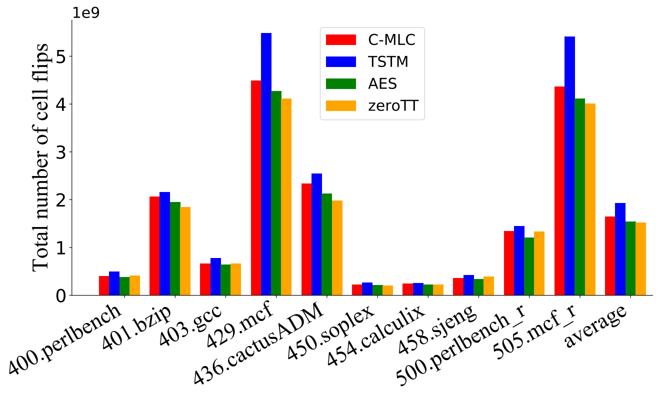


Figure 11: Total number of cell flips under four encoding methods.

In this evaluation, we use the total number of STT-RAM cell flips as a proxy to evaluate the endurance of different encoding schemes. We calculate the total number of cell flips by summing up the number of flips in the soft and hardware domain in STT-RAM cells. Fig. 11 shows that, compared with C-MLC STT-RAM, TSTM, and AES, zeroTT can also help improve the STT-RAM's endurance by reducing the total number of cell flips with an average of 7.9%, 21.2%, and 1.8%, respectively. Note that for some benchmarks, zeroTT experiences more cell flips due to the same reason in 4.3.

5 CONCLUSION AND FUTURE WORK

This paper presents zeroTT, a novel encoding scheme for MLC STT-RAM based cache, which can eliminate all the TTs in the STT-RAM cells during cache writes to improve the program performance, energy efficiency, and memory endurance. By analyzing and understanding the shortage of existing (2,3)-based coding methods, we proposed the new zeroTT, which is an optimal (3,4)-based expansion coding method. Our evaluation of SPEC CPU Benchmark Suites demonstrates that our zeroTT outperforms the state-of-the-art data encoding methods in access latency, energy consumption, and the total number of memory cell flips. Our future work will focus on two aspects. Firstly, We will enhance zeroTT to be able to dynamically adjust its coding method for the specific data patterns of an application. Secondly, we will further improve zeroTT's energy efficiency with an optimized written code selection algorithm.

ACKNOWLEDGEMENT

This work was supported by the Natural Science Foundation of Hunan Province of China (2021JJ10023).

REFERENCES

[1] Hooman Farkhani and Ali Peiravi. Low-energy write operation for 1t-1mtj stt-ram bitcells with negative bitline technique. *VLSI*, 24(4):1593–1597, 2016.

[2] Xunchao Chen, Navid Khoshavi, Jian Zhou, Dan Huang, Ronald F DeMara, Jun Wang, Wujie Wen, and Yiran Chen. Aos: Adaptive overwrite scheme for energy-efficient mlc stt-ram cache. In *DAC '16*, pages 170:1–170:6, 2016.

[3] Huizhang Luo, Jingtong Hu, Liang Shi, Chun Jason Xue, and Qingfeng Zhuge. Two-step state transition minimization for lifetime and performance improvement on mlc stt-ram. In *DAC '16*, pages 171:1–171:6, 2016.

[4] Jen-Wei Hsieh, Yi-Yu Liu, Hung-Tse Lee, and Tai Chang. Tse: Two-step elimination for mlc stt-ram last-level cache. *TC*, 70(9):1498–1510, 2020.

[5] Jen-Wei Hsieh, Yueh-Ting Hou, and Tai-Chieh Chang. Alternative encoding: A two-step transition reduction scheme for mlc stt-ram cache. *TCAD*, 41(8):2753–2757, 2021.

[6] Imtiaz Ahmad, Mahmoud Imdoukh, and Mohammad Gh Alfailakawi. Extending multi-level stt-mram cell lifetime by minimising two-step and hard state transitions in hot bits. *IET Computers & Digital Techniques*, 11(6):214–220, 2017.

[7] Jie Xu, Dan Feng, Wei Tong, and Jingning Liu. Encoding separately: An energy-efficient write scheme for mlc stt-ram. In *ICCD '17*, pages 581–584. IEEE, 2017.

[8] Xiangteng Zang, Xin Li, Yuqing Sun, Mengying Zhao, and Lei Dou. Energy optimization for multi-level cell stt-mram using state remapping. In *HPCC/SmartCity/DSS'16*, pages 546–553. IEEE, 2016.

[9] Xiaohua Lou, Zheng Gao, Dimitar V Dimitrov, and Michael X Tang. Demonstration of multilevel cell spin transfer switching in mgo magnetic tunnel junctions. *Applied Physics Letters*, 93(24), 2008.

[10] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[11] Seokin Hong, Jongmin Lee, and Soontae Kim. Ternary cache: Three-valued mlc stt-ram caches. In *ICCD '14*, pages 83–89. IEEE, 2014.

[12] Wang Kang, Weisheng Zhao, Zhaohao Wang, Yue Zhang, Jaques-Olivier Klein, Claude Chappert, Youguang Zhang, and Dafiné Ravelosona. Dfstt-mram: Dual functional stt-mram cell structure for reliability enhancement and 3-d mlc functionality. *IEEE transactions on magnetics*, 50(6):1–7, 2014.

[13] Wujie Wen, Yaojun Zhang, Mengjie Mao, and Yiran Chen. State-restrict mlc stt-ram designs for high-reliable high-performance memory system. In *DAC '14*, pages 1–6, 2014.

[14] Wei Zhao, Jie Xu, Xueliang Wei, Bing Wu, Chengning Wang, Weilin Zhu, Wei Tong, Dan Feng, and Jingning Liu. A low-latency and high-endurance mlc stt-mram-based cache system. *TCAD*, 42(1):122–135, 2022.

[15] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.