Tetris Write: Exploring More Write Parallelism Considering PCM Asymmetries

Zheng Li, Fang Wang[‡], Dan Feng, Yu Hua, Wei Tong[‡], Jingning Liu, and Xiang Liu
Wuhan National Lab for Optoelectronics, School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, China
Email:{lizheng, wangfang, dfeng, csyhua, Tongwei, jnliu, liuxiang0917}@hust.edu.cn

***Abstract*—The noises at the power lines limit the charge pump to provide large instantaneous current to PCM cells, which results in the number of bits can be written concurrently, i.e. the size of write unit, is restricted in PCM. When implementing PCM as the main memory, the inequality of cache line's size and write unit's size may result in many consecutive executed write units, which greatly decreases the system performance. Existing PCM write schemes, however, consider the worst power and time cases of written data, and ignore the actual current consumption. It is assumed that all data bits are changed and the electric current of each data unit is under fully utilized. The write performance is blocked due to pessimistic estimates, i.e. the current is often excessively supplied but is not used effectively, which leads to huge energy consumption. As a result, the write parallelism is limited and therefore restricts the overall system performance. To address this problem, this paper proposes a novel PCM write scheme named Tetris Write to explore more write parallelism and reduce the critical number of write units in PCM chip. The key idea behind Tetris Write is to monitor the number of '1' and '0' changed in each data unit, and schedule the order of data units' write-1 and write-0 execution considering not only the time and power asymmetries, but also the number asymmetry between RET and SET operations, to allow a larger number of concurrent bit-writes and make the best use of power supply. Tetris Write tries to schedule the dominating long term write-1s first and attempts to steal interspaces remained by write-1s to put the extraessential short write-0s. 4-core PARSEC benchmarks' results show that Tetris Write can get 65% read latency reduction, 40% write latency reduction, 46% running time reduction and 2X IPC improvement compared with the baseline on average. In addition, Tetris Write earns 26%, 15% and 10% more read latency reduction, 15%, 7% and 5% more write latency reduction, and outperforms 22%, 12% and 7% more running time reduction, compared with the state-of-the-art Flip-N-Write, 2-Stage-Write and Three-Stage-Write schemes, whose IPC improvements are 1.4X, 1.6X and 1.8X, respectively.**

***Index Terms*—PCM, write performance, asymmetries**

I. Introduction

Phase Change Memory (PCM) shows some outstanding features compared with DRAM, such as nonvolatile, no need to do refresh operations and smaller feature size while DRAM reaches its bottleneck in scalability without losing stability [1][2][3][4]. However, PCM shows some disadvantages in write performance and endurance [5][6][7]. Writing data into a PCM cell requires massive power but the noises at the power lines may limit the charge pump to provide large instantaneous current, which results in the number of bits can be written concurrently, i.e. the size of write unit, is restricted to a predefined constant [8][9]. Typical sizes of write unit are 8 ($X8$) and 16 ($X16$) bits per chip. Worse still, when use PCM in a mobile system, the amount of current that the system can provide may decrease and the number of cells that can be written concurrently must be reduced down to 4 and 2 bits to lessen the current consumption [8][10]. Due to the limited sizes of write unit, the write operation must be finished in write division mode and may need many sequentially executed write units. When implementing PCM as the main memory, the inequality of cache line's size and write unit's size may result in huge performance degradation [10]. For example, four 16 bits-width PCM chips consist of a memory bank and the write unit size is 16 bits, i.e. 2B per chip. In this circumstance, only 8B data can be written to a memory bank in parallel. A cache line is customarily 64B size, and it consumes ($\frac{64}{8} = 8$) write units to finish writing last level cache line down to PCM main memory [9][10][11][12][13][14][15][16][17][18]. In some latest enterprise processors and servers, the size of last level cache line is increased, e.g. 128B in IBM POWER7 [11][19] and 256B IBM zEnterprise [11][20]. With the increasing size of last level cache line, more write units are consumed and may cause significant system performance degradation.

Existing state-of-the-art PCM write schemes make a lot of efforts to solve the poor write performance problem. By exploring sample data compression, the maximum number of bits to be written is reduced and the size of write unit is doubled (Flip-N-Write [14]). By separating write-0 and write-1, write parallelism can be improved considering the power and time asymmetries (2-Stage-Write [11] and Three-Stage-Write [21]). However, they still consider the worst power and time cases when writing data, and ignore the actual current consumption of each data unit. It is assumed that all data bits are changed and the electric current of each data unit is under fully utilized. According to our experimental results, however, we observe that **(1)** the number of bits that are actually changed and consume power is not that much (9.6 bits per 64 bits data, only about 15%) and **(2)** the number of SET and RESET operations is unbalanced, and most workloads are SET-dominant. The write performance is blocked due to pessimistic estimates, i.e. the current is often excessively

[‡] Co-corresponding authors:
Fang Wang (wangfang@hust.edu.cn) and Wei Tong (Tongwei@hust.edu.cn)

supplied but is not used effectively. Low power utilization leads to huge power consumption, and limits the improvements in write parallelism and overall system performance.

In this paper, we propose a novel PCM write scheme named Tetris Write to improve the write parallelism and reduce the critical number of write units in PCM chip level. The key idea behind Tetris Write is to monitor the number of '1' and '0' changed in each data unit, and schedule the order of data units' write-1 and write-0 execution considering not only the time and power asymmetries, but also the number asymmetry between RET and SET operations, to allow a larger number of concurrent bit-writes and make the best use of power supply budget. Tetris Write tries to schedule the dominating long term write-1s first and attempts to steal interspaces remained by write-1s to put the extraessential short write-0s. Tetris Write divides the write into three stages: **read, analysis** and **individually write**.

- In read stage, Tetris Write reads out the original data first and flips the data if more than half of bits have to be changed like Flip-N-Write. Thus, no more than half of total bits need to be written. Moreover, the actual number of SET and RESET operations per data unit is also calculated.
- In analysis stage, Tetris Write calculates the power needs of write-1 and write-0 considering the power asymmetry based on the results of read process. Then Tetris Write schedules the order of data units' write-1 and write-0 executions considering the number and time asymmetries.
- In individually write stage, the write-0 and write-1 execute individually and simultaneously under the careful control of FSMs.

The remainder of this paper is structured as follows. Section II describes the background and summaries related work. Section III presents the motivations and key ideas of Tetris Write. Section IV introduces the architecture and implementation of Tetris Write scheme. Section V presents and analyzes the experimental results. Finally, section VI offers conclusions.

II. Background and Related Work

Phase Change Memory is a type of Non-Volatile Memory (NVM) and consist of chalcogenide glass material, such as Ge2Sb2Te5 (GST). GST shows the unique behavior in resistance level under amorphous state and crystalline state. In general, the resistance level of amorphous state and crystalline state differs ten or more order of magnitudes, which can be used for storing digital information '1' and '0'. A PCM cell can store one or more than one bit, called single-level-cell (SLC) and multiple-level-cell (MLC), respectively. Typical PCM cell structure and the write and read processes are illustrated in Figure 1. A PCM cell is simply composed of top and bottom electrodes, heater and GST material. PCM shows both time and power asymmetries of writing '1' (SET) and '0' (RESET) [11][22]. In general, making the GST to the amorphous phase (RESET) is typically achieved in less time but needs higher current, whereas switching GST to the crystalline state (SET) consumes longer time but requires

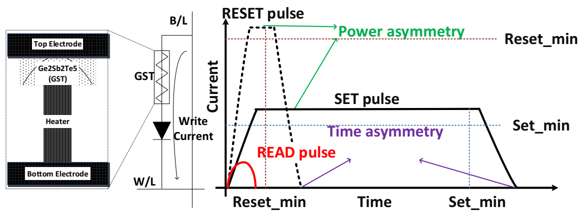

Fig. 1: PCM cell structure and the operations process.

lower current. In other words, RESET is fast but consumes more current while SET is slow and needs lower instantaneous current. According to existing literature, T_{set} is about 8 times longer than T_{reset} while current consumption C_{set} is about half of C_{reset} [8][9]. Moreover, by applying low voltages, the information in a PCM cell is read by judging the current level, whose value is decided by the resistance level of the GST material. In this study, we focus on SLC PCM for its better write performance. In addition, the noises at the power lines may limit the charge pump in chip to provide large instantaneous current, which results in the number of bits can be written in parallel, i.e. the size of write unit, is restricted in PCM chip. Typical sizes of write unit are 8 and 16 bits per chip [10]. Differently, the current need of a read operation is much lower than a SET or RESET operation and it is feasible to read out the values among hundreds of PCM cells concurrently.

The main memory is usually organized as a hierarchical architecture [13], as shown in Figure 2. The main memory consists of many memory ranks, which are composed by several memory banks. In order to match the data bus width, many PCM chips are used and compose a memory bank. The inequality of cache line's size and write unit's size may result in many consecutive executed write units to finish a cache line write service, which greatly decreases the system performance. Four 16 bits-width PCM chips compose a memory bank and the write unit size is 16 bits, i.e. 2B. In this circumstance, only 8B data can be written in parallel to a memory bank. The cache line is typical 64B size and it needs 8 write units to finish the write service in PCM main memory.

Conservative conventional write scheme considers the worst current requirements (RESET) and time requirements (SET) of PCM write, and it often consumes many consecutive write units to finish a cache line write service, regardless of the actual data content. Assuming the size of write unit is M bytes and the size of cache line is N bytes, the time consumption for writing a cache line data is concluded in Equation 1.

$$T_{Conventional} = \frac{N}{M} \times T_{set} \tag{1}$$

Flip-N-Write (FNW) [14] flips the data when the hamming distance between original and new data is more than half of

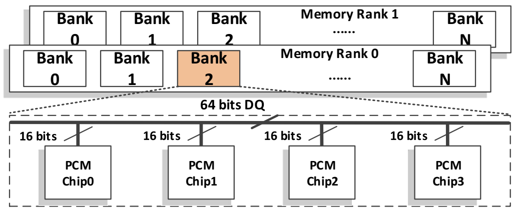

Fig. 2: Hierarchical Architecture of Main Memory

data width. Thus, two data units can be written in one write unit under the power constraints. Flip-N-Write doubles the size of write unit and reduces the time consumption of write service. Compared with the conventional write scheme, Flip-N-Write introduces an extra read operation and the average service time is concluded in Equation 2.

$$T_{Flip-N-Write} = T_{read} + \frac{1}{2} \times \frac{N}{M} \times T_{set} \tag{2}$$

2-Stage-Write [11] tries to improve the write performance by exploiting the power and time asymmetries of RESET and SET. 2-Stage-Write divides write into two consecutive processes: stage-0 and stage-1. In stage-0, RESETs in all data units are executed first. Since T_{reset} is much shorter than T_{set}, stage-0 finishes within a short time. In stage-1, SETs in two data units can be executed together for the current consumption of SET C_{set} is half of RESET C_{reset}. By flipping the write data if more than half of SETs in a write operation, the execution speed of stage-1 is doubled again. Assuming the time ratio of SET and RESET is K and the power ratio is $1/L$, we can conclude the time consumption in Equation 3.

$$T_{2StageWrite} = (\frac{1}{K} + \frac{1}{2L}) \times \frac{N}{M} \times T_{set} \tag{3}$$

Three-Stage-Write or 3-Stage-Write [21] tries to combine Flip-N-Write and 2-Stage-Write in effective ways. Compared with 2-Stage-Write, Three-Stage-Write adopts an extra process before stage-0 and stage-1. Three-Stage-Write first reads the original data and determines whether flips the data or not similar to Flip-N-Write. Three-Stage-Write can double the execution speed of stage-0 and stage-1. Compared with the 2-Stage-Write, the overall waiting time of stage-1 is same while the waiting time of stage-0 in cut by half. Three-Stage-Write also need one read operation before writing and the average service time is shown in Equation 4.

$$T_{3StageWrite} = T_{read} + (\frac{1}{2K} + \frac{1}{2L}) \times \frac{N}{M} \times T_{set} \tag{4}$$

III. Motivation and Tetris Write

A. Observation and Motivation

The experimental result of the number of RESET and SET operations per data unit, i.e. 64 bits, is shown in Figure 3. The experimental settings and workload characteristics are described in Table II and III. It is worth noticing that this number is counted after the data inversion processing adopted by Flip-N-Write and Three-Stage-Write, i.e. the data are inverted if more than half of bits need to be executed RESET and SET operations. We draw two critical observations that:

- **Observation 1:** The number of total bit-write operations, including RESET and SET, is not that huge in one data unit on average. Based on our experimental result, only a small part of the bits are need to be changed and consume power. On average, the number of bit operations is 9.6 per 64 bits write (about 15%), including 2.9 RESET and 6.7 SET operations.
- **Observation 2:** RESET/SET operations are heterogeneous among workloads and inside one workload. On the one hand, the number of write-1 (SET) and write-0 (RESET) varies much in different workloads. For example, there are only 2 bit-writes in blackscholes while vips consumes 19 bit-writes on average. On the other hand, most workloads are SET-dominant while few are fifty-fifty SET/RESET, such as vips and ferret.

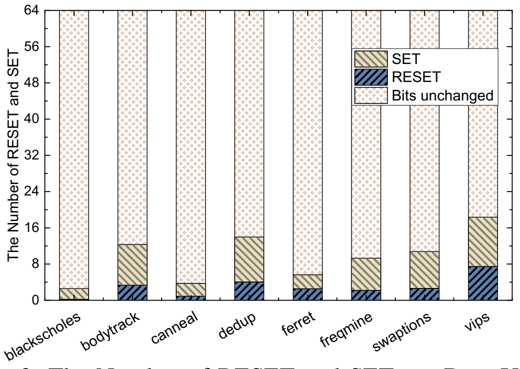

Fig. 3: The Number of RESET and SET per Data Unit

State-of-the-art schemes, however, consider the worst power and time cases when writing down data, and ignore the circumstances we observed. They all assume that all data bits are changed and the electric current is fully utilized, and may led to significant wastage of power budget, i.e. the power supply. On the one hand, regardless of conventional write scheme and Flip-N-Write, they both regard RESET and SET in the same way and consider the write in the worst power consumption case. However, based on our **Observation 1**, there are few bit operations. Although Flip-N-Write performs two data units concurrently, it still suffers from low utilization of power budget ($(9.6 \times 2) \div 64 \approx 30\%$) and huge consumption of power budget supply. On the other hand, even though 2-Stage-Write and Three-Stage-Write leverage the asymmetries and improve the write parallelism, they still consider the write in the worst power consumption case and unconcern the actual current utilization in stage-0 and stage-1. However, based on our **Observation 2**, most workloads are SET-dominant with few RESET operations, and the stage-0 may earn extremely low power budget utilization. The current is often excessively supplied without being used effectively. Low power budget utilization limits improvements in write parallelism and overall system performance.

B. Tetris Write

Unlike conventional, Flip-N-Write, 2-Stage-Write and Three-Stage-Write, Tetris Write tries to take into account the actual current consumptions of every data units when writing '0' and '1', respectively, and schedules the order of data units' execution considering not only the time and power asymmetries, but also the number asymmetry between RET and SET operations, to make full use of the power budget. Tetris Write divides the write into three stages: read, analysis and individually write.

1) ***Read:*** The working flow of the read process is shown in Algorithm 1. Tetris Write leverages the read-before-write scheme to reduce the data amount. Tetris Write first reads out the original data and its flip tag {D', F'}, and flips the

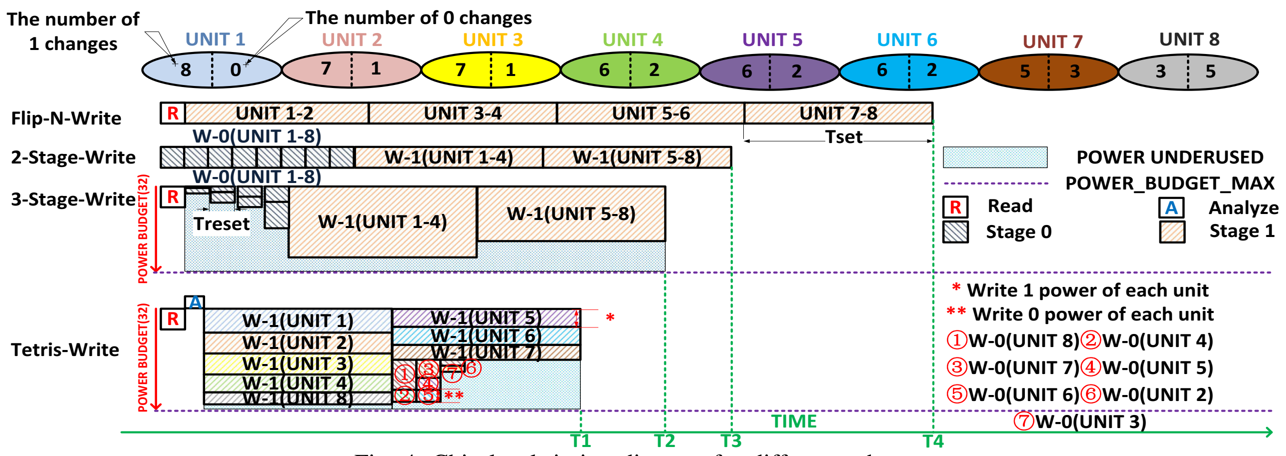

Fig. 4: Chip-level timing diagram for different schemes

data if the hamming distance between original data {D', F'} and new data {D, 0} is more than half of the write unit size. Furthermore, Tetris Write monitors and records the number of write-1 bits and write-0 bits after data inversion.

Algorithm 1 The read process of Tetris Write

```
Require:
    /*A is the write address; N is the write unit size in bits*/
    /*F' is the flip tag of original data; F is the flip tag of new data*/
    /*D is the data unit to be written; D' is the original data*/
Ensure:
1: $F' = Read\_the\_flip\_bit(A)$
2: $D' = Read\_the\_data\_bits(A)$
3: if $Hamming\_dist(\{D,0\},\{D',F'\}) > N/2$ then
4:     $D =\sim D, F = 1;$
5: else
6:     $D = D, F = 0;$
7: end if
8: $N1 = Count\_the\_number\_of\_1(D)$
9: $N0 = Count\_the\_number\_of\_0(D)$
```

2) ***Analysis:*** Tetris Write separates the write-0 and write-1 of each data units and considers the execution sequence of them respectively. In analysis stage, Tetris Write receives the number of write-1 and write-0, and calculates the current need of write-1 and write-0 each data unit considering the power asymmetry of PCM. In general, based on **Observation 2** and time asymmetry, Tetris Write tries to schedule the dominant and long-term write-1 stage of every write units. After that, Tetris Write attempts to schedule all data units' write-0s together with the execution of write-1s under the power constraints. Thus, the long write-1 stage can hide the fast write-0 and the number of write units can be reduced. In other words, Tetris first determines the write-1 scheduling scheme considering the current need of all data units when writing '1', then tries to put all write-0 in the interspace remained by write-1s and get the least number of write units

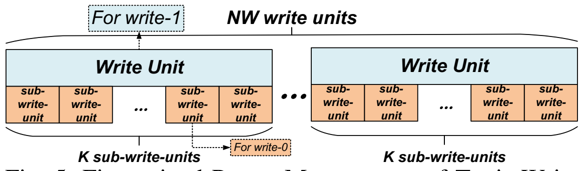

Fig. 5: Fine-gained Power Management of Tetris Write

like the classical Tetris game.

The pseudocode of analysis process is shown in Algorithm 2. Tetris Write is a greedy algorithm and the key idea is to find the first-fit write unit for each data unit. To provide fine-gained current management, one write unit is divided in K sub-write-units considering the time asymmetry of SET and RESET as shown in Figure 5. NW is the number of existing write units. In general, the power management of write-1s is write unit level and is sub-write-unit level for write-0s. Another thing to note is that write unit together with sub-write-unit should follow the power constraints.

Tetris Write first calculate the current consumption of write-1 and write-0 of every data units considering the power asymmetry. Then, Tetris Write sorts all write-1 data units in the decreasing order according to the current need of write-1s ($IN1[i]$). Assuming each sub write unit has currently used energy $WU_p[j]$ and the maximum power budget is PB_{max}. The initial values of $WU_p[j]$ are '0'. First, Tetris tries to put write-1 of $dataunit[x]$ that consumes most power into $writeunit[1]$. If $IN1[x] + WU_p[1] < PB_{max}$, marking that data unit should be executed in write unit '1' and update the power consumption values of $WU_p[1]$ ($WU_p[1] = WU_p[1] + IN1[x]$). Second, Tetris tries to put the most power-cost $dataunit[y]$ into existing write unit, i.e. $writeunit[1]$ now, if $IN1[y] + WU_p[1] > PB_{max}$, then we need another write unit, i.e. $writeunit[2]$ now and mark the data unit with tag $writeunit[2]$. Universally, for each $dataunit$, in turn examines whether existing $writeunit[1 \to j]$ can handle the data unit, if not, employ another write unit to handle the data unit. After all write-1 data units have been scheduled, Tetris arranges all write-0 data units similar to write-1 data units. Differently, the initial values of $WU_p[j]$ are not zero but the remaining values after write-1 processing. If all existing $writeunits$ can't meet the demand of write-0, Tetris Write uses an additional $subwriteunit$ to finish the write-0.

In analysis stage, all data unit's execution sequence of write-1 and write-0 is determined and saved in two individual queues, write-1 and write-0 queue, respectively. And all write-1 and write-0 are labeled with the write unit they belong to.

*3) **Individually Write:*** In individually write stage, two Finite State Machines, FSM0 and FSM1, generate the data units select signals and write signals and send them to the write driver, according to the schedule results of write-1 and write-0 of all data units, which are stored in write-1 and write-0 queues. The FSMs are driven by the memory clock and leverages internal counters to achieve precise control of execution of write-1 and write-0. Write driver receives the write data and write signals (SET/RESET) from FSMs, and generates the PROG enable and SET/RESET enable to the corresponding PCM cells.

Algorithm 2 Tetris Write Algorithm

```
Require:
    The time asymmetry is $K$; The power asymmetry is $L$;
    The number of data unit, $NW$; The max power budget, $PB_{max}$;
    The number of write-1 data unit, $NUM1[NW]$;
    The number of write-0 data unit, $NUM0[NW]$;
    Current each sub-write-unit's power use, $WU_p[NW \times K]$;
Ensure:
    The number of write units for all write-1s, $result$;
    Additional sub-write-units for write-0s, $sub-result$;
 1: /*Get write-1&write-0 current need considering asymmetry*/
 2: for each $i \in [1, NW]$ do
 3:     $IN1[i] \leftarrow NUM1[i]$;
 4:     $IN0[i] \leftarrow NUM0[i] \times L$;
 5: end for
 6: $result \leftarrow 1$; $subresult \leftarrow 0$;
 7: for each $i \in [1, NW]$ do
 8:     Sort each $IN1[i]$ in the decreasing order;
 9:     Sort each $IN0[i]$ in the decreasing order;
10: end for
11: /*Traverse all write-1 data units*/
12: for $i \leftarrow (1 \rightarrow NW)$ do
13:     /*Find whether existing write units can satisfy it*/
14:     for $j \leftarrow (1 \rightarrow result)$ do
15:         /*The time asymmetry is K*/
16:         if $(IN1[i] + WU_p[j \times K]) > PB_{max}$ then
17:             /*If existing write units can't meet, need another one*/
18:             if $j = (result - 1)$ then
19:                 $result \leftarrow result + 1$;
20:             else
21:                 Send data unit $i$ to write-1 queue with tag $j$;
22:                 /*One write-1 occupies $K$ sub-write-unit*/
23:                 for $k \leftarrow (1 \rightarrow j \times K)$ do
24:                     $WU_p[k] \leftarrow (WU_p[k] + IN1[i])$;
25:                 end for
                    $break$;
26:             end if
27:         end if
28:     end for
29: end for
30: /*Traverse all write-0 data units*/
31: for $i \leftarrow (1 \rightarrow NW)$ do
32:     /*Find whether existing sub-write-units can satisfy it*/
33:     for $j \leftarrow (1 \rightarrow result \times K)$ do
34:         /*If existing can't meet, need subwriteunit for write-0*/
35:         if $(IN0[i] + WU_p[j]) > PB_{max}$ then
36:             if $j = (result - 1)$ then
37:                 $subresult \leftarrow subresult + 1$;
38:             else
39:                 $WU_p[j] \leftarrow (WU_p[j] + IN0[i])$;
40:                 Send data unit $i$ to write-0 queue with tag $j$;
                    $break$;
41:             end if
42:         end if
43:     end for
44: end for
```

The write service time of Tetris Write can be expressed by Equation 5. The performance is decided by three parameters, i.e. $result$, $subresult$ and K. K is the time ratio of SET and RESET while $result$ and $subresult$ are respectively the number of write units for write-1s and write-0s. The values of $result$ and $subresult$ are calculated in Algorithm 2.

$$T_{TetrisWrite} = (result + \frac{subresult}{K}) \times T_{set} \tag{5}$$

Figure 4 is a simple sample to emphasize the significantly differences between Tetris Write and the state-of-the-art PCM write schemes, e.g. Flip-N-Write, 2-Stage-Write and Three-Stage-Write. Assuming cache line size is 64B, four chips under $X16$ write division mode form a memory bank and original size of write unit is 8B per bank, 2B per chip. The power budget is set to 128 per bank and 32 per chip under the assumption that the power ratio between RESET and SET is 2. In other words, 32 SET and 16 RESET operations can be operated concurrently per chip, i.e. 128 SET and 64 RESET per bank. Under the conventional scheme, each write unit completes after a service time required writing a one, regardless of the actual values written. The conventional scheme requires eight write units serially, i.e. $8T_{set}$ in total. Considering that is quite a long time, we don't show it in Figure 4. Flip-N-Write first read original data and compares it against new data, and then effectively reduces the number of bits to be written by half of the data unit size, and thus it allows writing two write units concurrently under the power constraints. Flip-N-Write completes the write of a cache line at T_4. In 2-Stage-Write, all zeros are written to PCM cells at a fast speed. During the following write-1 stage, the lower current requirement of writing '1' enables more ones to be written concurrently under the same power constraints. Therefore, the number of serial writes in the write-1 is reduced, which leads to the completion time to be T_3. But things become different when it comes to Three-Stage-Write. The 0-bits and 1-bits actually changed both reduce by half. Because of the data amount reduction of stage-0 and stage-1, the stage-0 of three-stage-write takes less time, completion time is thus changed to T_2.

Like Flip-N-Write and Three-Stage-Write, Tetris Write needs to read data first, then we have the analysis stage to determine the scheduling scheme of write-1 and write-0 of all data units. Since Tetris Write attempts to schedule all data units' write-0 together with the execution of write-1 under the power constraints, the long write-1 stage can hide the fast write-0 stage and the number of write units can be further reduced. In the specific circumstance given in Figure 4, write-1s of $dataunit[1-4]$ and $dataunit[8]$ can be written in parallel under the power constraints $(8+7+7+6+3 = 31 < 32)$. Moreover, when write-1s of $dataunit[5-7]$ is under service, write-0s of $dataunit[8]$ and $dataunit[4]$ $((6+6+5)+2\times(5+2) = 31 < 32)$, $dataunit[5-7]$ $((6+6+5)+2\times(3+2+2) = 31 < 32)$ and $dataunit[2-3]$ $((6+6+5)+2\times(1+1) = 21 < 32)$ can be served in a

TABLE I: Differences of Various Write Schemes

Scheme	Key Idea	Reduce Latency	Reduce Energy
Flip-N-Write	Reducing the bits to be written by encoding the data with extra flip-bit	YES	YES
2-Stage-Write	Asymmetries of PCM, divide PCM write into two stages considering the power and time asymmetries	YES	NO
Three-Stage-Write	Asymmetries of PCM, similar to 2-Stage-Write, add read operation before write to speed up stage-0	YES	YES
Tetris Write	Rescheduling the execution of write-1 and write-0 considering the actual current consumption	YES	YES

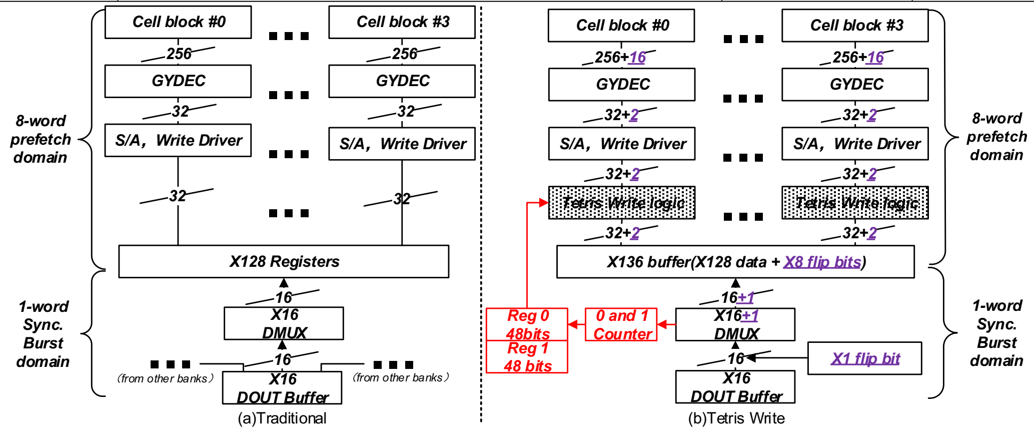

Fig. 6: Data Path

row considering the power and time asymmetries. Tetris Write completes the service most quickly, i.e. T_1.

Table I shows the comparisons between Tetris Write and existing state-of-the-art PCM write schemes. Flip-N-Write's key idea is reducing the bits to be written by flipping the data or not, it can reduce write latency and energy consumption. 2-Stage-Write divides PCM write into two stages considering the power and time asymmetries. It can also reduce write latency, however, it cannot reduce energy consumption for it doesn't reduce the overall data amount. Three-Stage-Write makes use of the same principle as 2-Stage-Write, but it adds extra data operation before write stages like Flip-N-Write and therefore speeds up write-0 stage. It can reduce both write latency and energy consumption. Compared with the state-of-the-art PCM write schemes mentioned above, Tetris Write schedules the execution of write-1 and write-0 based on the actual current consumptions. By introducing the read-before-write scheme similar to Flip-N-Write, the data amount is reduced and our scheme can also reduce write latency and energy consumptions.

IV. Implementation

Our implementation of Tetris Write is based on an industrial prototype from Samsung [8][9]. By introducing a buffer for read requests, the original design support high-speed synchronous read operation and 128 bits data can be read out under the synchronous read operation mode. However, the size of write unit is limited to no more than 16 bits for one chip. What's more, the original design does not notice the power and time asymmetries, it considers the finish time of one data unit to be T_{set}. Moreover, in order to avoid imbalanced current demand caused by uneven cache line data distribution, we adopt the global charge pump (GCP) technology provided in Ref. [16]. GCP uses a global bridge chip and dedicated wires among PCM chips. Thus, a PCM chip can "steal" current from other PCM chips and the size of write unit per bank is stable [16], i.e. 64B in our study.

A. Datapath

Figure 6 shows the comparison of Tetris Write's datapath and conventional datapath. Cell blocks refer to the PCM array, a PCM chip customarily consists of multiple cell blocks. GYDEC refers to global column decoder which selects a column in the PCM array. S/A refers to sense amplifier, DMUX refers to multiplexer and DOUT refers to data out buffer, respectively. Our design adds extra write logic named Tetris Write Logic in order to implement the processes of read and analysis. Furthermore, in order to provide data inversion support, the data width is extended, as the underlined part in Figure 6(b). 0/1 counters are also added to monitor the number of 0/1 that need to be written. Reg0 and Reg1 are two 48 bits long arrays used to store the label of data units and the number of 0/1, respectively. There are 8 data units and the number of bits changed is less than half of chip's data width, i.e. 8 bits. So we need 6 bits to store one data unit's label and it's actual number of 0 or 1 in Reg0 or Reg1. The read data may pass through cell blocks, GYDEC, S/A and DOUT buffer successively while the write data pass through write buffer, Tetris Write Logic, S/A write driver, GYDEC and cell blocks in turn. It is worth noting that the additional logic layer doesn't add any extra overhead on the read operation, which is on the critical path of the system performance [23][24].

B. Tetris Write Logic

Tetris Write Logic is indicated by the dashed box in Figure 7. Reg0 and Reg1 store numbers of 0/1 for each data unit and

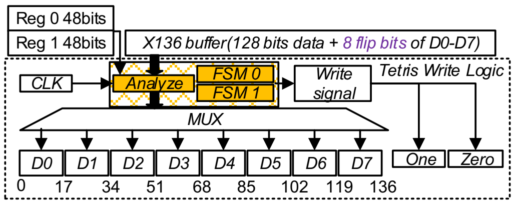

Fig. 7: Write Control Logic

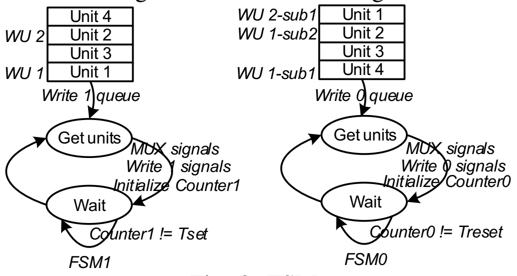

Fig. 8: FSMs

the serial number of each data unit as mentioned in datapath part. CLK refers to memory bus clock. MUX is a multiplexer, according to fixed offset to select the appropriate data unit. When performing a write on PCM, Tetris-Write logic receives the data from the write buffer, the number of 0/1 for each data unit stored in Reg0 and Reg1, then sends them to the analyzer. The analyzer determines the write-1 and write-0 execution order of every data unit and sends them to the two individual queues, i.e. write-1 queue and write-0 queue. Figure 8 shows the detailed design of FSMs, the jobs of FSM0 and FSM1 are generating the data units select signals and write signals, and sending them to the write driver. For example, FSM1 first gets the serial number of data units that to be executed and sends the data units select signals to MUX. After T_{set}, FSM1 checks and retrieves the next write-1 data units and repeats the whole process. Similarly, FSM0 periodic retrieves the data units under write-0 in write-0 queue every T_{reset}. It is worth noting that FSM0 and FSM1 are independent of each other and the executions are simultaneous.

C. Write driver

In order to support the proposed write scheme, we redesign the write driver as shown in Figure 9. Write driver receives the write signals produced by the FSMs to determine the SET/RESET signals. Because Tetris write only writes the bits need to changed, a XOR gate is used for different bits judgment. Meanwhile, an extra "PROG enable" signal is also needed similar to Flip-N-Write. The PROG enable signals are produced by the XOR gate mentioned above, when a bit of old data comes from read buffer differ from the new data comes from DX, the corresponding bit of PROG enable signals

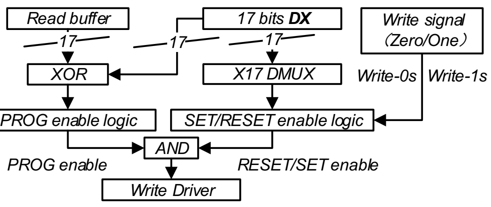

Fig. 9: Write Driver

is set to one. In conventional PCM write driver, only the SET/RESET enable signals are needed, but in our design, data are written only when SET/RESET enable signal and PROG enable signal are both active. For example, assume that the PROG enable signal of a certain bit is '0', which means this bit doesn't need to be changed. And it's SET/RESET signal is 'SET'. When these two signals come to write driver together, it won't perform SET operation to that bit.

D. Overhead

In this discussion, we analyze the sources of extra overheads and their impact in terms of performance in detail. We focus on the overheads due to read process and analysis process.

On the one hand, similar to Flip-N-Write and Three-Stage-Write, Tetris Write flips the data unit if more than half of bits need to do RESET and SET operations. This data inversion reduces the data amount but introduces an independent read operation before writing. The overhead can be considered as T_{read}, which is 50ns in our study.

On the other hand, the Tetris Write algorithm in the analysis process may deliver extra overhead including a small number of sort operations (only 8 write-1 and write-0 in this study) and rearrangement of these write-1 and write-0 units, and we name this overhead as analysis overhead. In order to accurately measure the time of analysis overhead, we try to implement the Tetris Write algorithm in FPGA using Verilog HDL. We use Vivado HLS tools and implement the Tetris Write algorithm in Xilinx Vitex-7 FPGA (xc7vx330tffg1157). Vivado HLS is a high-level synthesis tools provided by Xilinx and enables C, C++ and System C programs to be directly targeted into Xilinx devices without the need to manually create RTL [25]. Based on the experimental results of Vivado HLS, the Tetris Write algorithm consumes 41 cycles under the worst case running at 400 MHz clock, the same clock frequency as memory bus clock. However, our assumptions of clock is based on the existing literature [14] and the measurement of analysis overhead is primitive and pessimistic. We can shorten the analysis time by migrating the work to an ASIC with individual clocks with higher frequency.

Tetris Write improves write parallelism by adding individual Tetris Write logic, and added circuits may introduce extra area and power overhead. In general, write is not on the critical path of performance. Tetris Write extends the write datapath, however, the critical read datapath stays the same. In addition, the added write control logic is much less complicated compared with existing cost-sensitive components in PCM, such as the charge pumps, program-and-verification circuits for write control etc. [11]. Tetris Write also changes the write driver slightly and writes different bits, which can be easily implemented with XOR gate and a counter. Compared with the original write driver, "SET/RESET" signals for one bit are AND-gated with the "PROG enable" signal. Considering XOR and AND logic gates is relatively simple, the area overhead hence is minimal. The extra power consumption is less than 4 mW according to the results of VIVADO tools. The output node of the pump is about 5V in write mode, draws about

25 mA current in division-write mode. Thus write consumes $5 \times 25 = 125$ mW power in the baseline prototype [9]. The power overhead is acceptable ($\frac{4}{125} \approx 3.2\%$).

V. Evaluation

A. Experimental Setting

We use the event-driven GEM5 simulator to simulate a multi-core computer system running with multi-threaded PARSEC 2.0 benchmark suite [26] and evaluate our proposed PCM write scheme. We add individual NVM module design based on NVmain [27] on the GEM5 simulator [28]. The multi-threaded PARSEC2.0 benchmarks we used were present and profiled in Ref. [29]. To test the effectiveness of Tetris Write design, we use 8 multi-threaded workloads from PARSEC 2.0 benchmark suite. All these workloads are taken from various real-world applications majoring in Financial Analysis, Computer Vision, Engineering, Enterprise Storage etc.. The memory Read Per Kilo Instructions (RPKI), memory Write Per Kilo Instructions (WPKI) are also summarized in Table III. The system baseline configuration is shown in Table II. The parameters of main memory are taken from PCM hardware prototype published by Samsung [9]. We use ALPHA-like processors with 4 O3 (out-of-order) cores, whose frequency is 2GHz. Moreover, there is 3-level cache architecture and the size of last level DRAM cache is 32MB. The memory controller is the same with previous work [16]. It takes 50ns, 53ns, 430ns to read a PCM cell, RESET a cell and SET a cell, respectively. Moreover, we assume the ratio of current RESET to SET is 2 and the original write unit size is 8B per memory bank. The cache line size is 64B, and the number of write unit under baseline is 8.

We try to discuss the impact of PCM systems adopting Tetris Write based on some critical measure of system performance, such as read latency, write latency, IPC and applications' running time. In our study, we adopt the DCW scheme [30] as the baseline and try to compare Tetris Write with the state-of-the-art PCM write schemes, such as Flip-N-Write [14], 2-Stage-Write [11] and Three-Stage-Write [21].

B. Experimental results

*1) **The Number of Write Units**:* Since sequentially executed write units are the primary cause of poor write performance, the number of write units is a very critical indicator for performance evaluation. Figure 10 shows the number of sequential write units of Tetris Write on average compared with theoretical values of state-of-the-art schemes. We get several observations from the results. First, in all benchmarks, Tetris Write gets better number of write units compared with the state-of-the-art write schemes. The write operations can be finished with fewer sequentially executed write units and the overall system performance can be greatly improved. On average, Tetris Write needs 1.06 to 1.46 write units to finish the cache line write service. In comparison, Flip-N-Write doubles the size of write unit with data inversion and it usually takes 4 write units. 2-Stage-Write needs 3 write units, 1 for stage-0 and 2 for stage-1. Three-Stage-Write doubles the speed of stage-0, we can consider it uses 2.5 write units to finish the cache line service. Second, the number of write units of Tetris Write varies much among different workloads. Tetris Write shows significant improvement over the state-of-the-art schemes when there are few RESET and SET operations, such as blackscholes, canneal and ferret. However, when there includes many RESET and SET operations in workloads, such as dedup and vips, Tetris Write doesn't reduce the number of write units distinctly.

TABLE II: Parameters of Simulation

Parameter	Value
CPU	4-Core CMP, 2GHz, ALPHA-architecture processor with O3 (out-of-order) cores.
Cache Organization	64-byte line size
L1 Cache	32KB I-cache, 32KB D-cache, 2 cycles access latency
L2 Cache	8-way, 2MB, 20 cycles access latency
L3 Cache	16-way, 32MB, 50 cycles access latency
Memory Controller	FRFCFS scheduling, 32-entry R/W queues
Memory Organization	4GB SLC PCM, 64 bits data width, single-rank, 8 banks
PCM Organization	4-$X16$ PCM chips, 8B write unit size
Memory Timing	READ 50ns, RESET 53ns, SET430 ns
Memory Energy	Ratio of RESET and SET current is 2

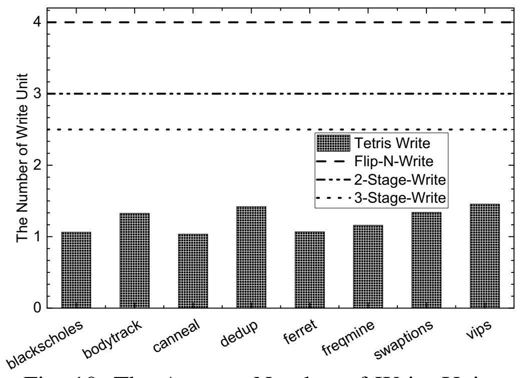

Fig. 10: The Average Number of Write Units

*2) **Read Latency**:* Figure 11 presents the average read latency of memory requests Tetris Write and the state-of-the-art write schemes compared with the baseline. Tetris Write achieves 65% read latency reduction compared with the DCW on average while Flip-N-Write, 2-Stage-Write and Three-Stage-Write shows only 39%, 50% and 56%, respectively. In addition, Tetris Write outperforms mentioned PCM write schemes in all workloads. Three of eight workloads (blackscholes, ferret and freqmine) show more than 10% read latency improvement over the Three-Stage-Write, whose performance is on the top of the existing PCM write schemes. Moreover, dedup and vips also shows better read latency reduction over Flip-N-Write, 2-Stage-Write and Three-Stage-Write, even one data unit may include many RESET and SET operations. As Tetris Write reduces the write service time and read requests benefit from the waiting time reduction, the overall read latency can be shortened. Read is on the critical path of the main memory systems, therefore, Tetris Write can successfully improve the performance of the main memory system.

TABLE III: Multi-threaded Workloads Used in Our Study

Program	Application Domain	Data Usage of Sharing	Data Usage of Exchange	RPKI	WPKI
blackscholes	Financial Analysis	low	low	0.04	0.02
bodytrack	Computer Vision	high	medium	0.72	0.24
canneal	Engineering	high	high	2.76	0.19
dedup	Enterprise Storage	high	high	0.82	0.49
ferret	Similarity Search	high	high	1.67	0.95
freqmine	Data Mining	high	medium	0.62	0.25
swaptions	Financial Analysis	low	low	0.04	0.02
vips	Media Processing	low	medium	2.56	1.56

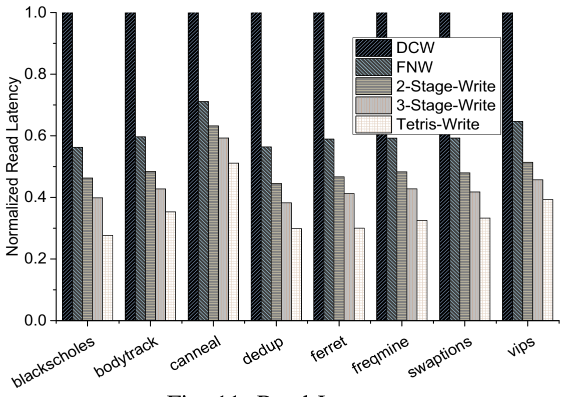

Fig. 11: Read Latency

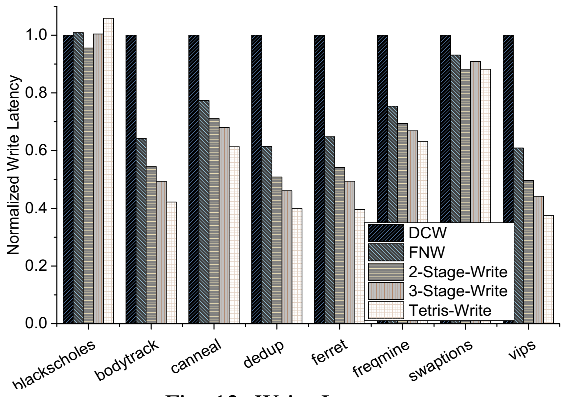

Fig. 12: Write Latency

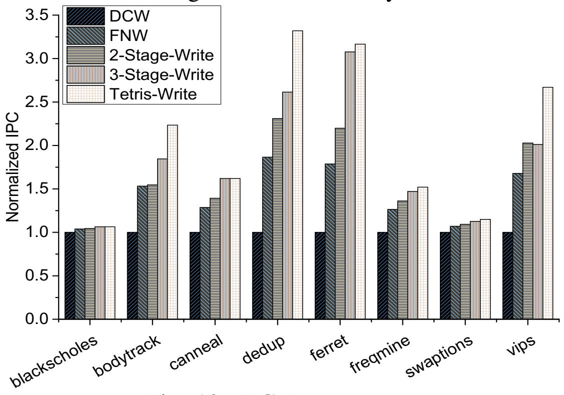

Fig. 13: IPC Improvement

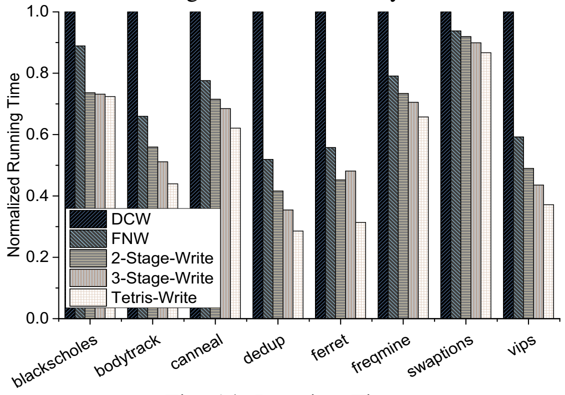

Fig. 14: Running Time

*3) **Write Latency**:* The experimental results of write latency are shown in Figure 12. Compared with the baseline, Tetris Write reduces more than 40% write latency. Moreover, Tetris Write outperforms 15%, 7% and 5% more write latency reduction compared with Flip-N-Write, 2-Stage-Write and Three-Stage-Write, respectively. It's worth noting that in blackscholes and swaptions, the improvement in write latency of Tetris Write is not that obvious, yet we get write performance degradation. There are two reasons for this phenomenon. On the one hand, the memory controller we adopted has individual write and read queues. The variable FRFCFS scheme schedules the read request first and services the write requests only when the write queue is full. However, both blackscholes and swaptions are read-dominant workloads, the number of write operations is far behind read operations. So the waiting time for a write request is quite long, which results in long service time of a write. On the other hand, the implementation of Tetris Write incurs extra overhead in analysis process. Besides blackscholes and swaptions, other workloads show much better write latency reduction compared with Flip-N-Write, 2-Stage-Write and Three-Stage-Write, respectively.

*4) **IPC**:* As Tetris Write can reduce the service time of a cache line write, IPC may benefit from the system performance improvement. We use IPC improvement as defined in Equation 6 to measure the IPC of all compared schemes. Figure 13 shows the IPC improvement for all workloads.

$$IPCImprovement = \frac{IPC}{IPC_{baseline}} \qquad (6)$$

Tetris Write shows maximum IPC improvement than other state-of-the-art schemes under all workloads, compared with the baseline. On average, Tetris Write gets 2X IPC improvement compared with the baseline while Flip-N-Write, 2-Stage-Write and Three-Stage-Write earn 1.4X, 1.6X and 1.8X IPC improvements, respectively. This can be explained by the low read latency of Tetris Write, which means more instructions can be executed in the same period of time.

*5) **Running Time:*** The running time of applications is shown in Figure 14. And we try to compare Tetris Write with the state-of-the-art PCM write schemes. Tetris Write reduces the write service time of a cache line downtown PCM main memory, which can reduce the read latency and is directly mapped into applications' running time. Under 8 multi-threaded benchmarks in this study, Tetris Write outperforms all compared write schemes. Tetris Write earns more than 46% running time reduction compared with baseline on average. Meanwhile, Tetris Write outperforms 22%, 12% and 7% more running time reduction, compared with Flip-N-Write, 2-Stage-Write and Three-Stage-Write, respectively.

VI. Conclusion

This paper proposes a novel PCM write scheme named Tetris Write to improve the write parallelism and reduce the write service time of PCM. Tetris Write explores both the time and power asymmetries of writing '1' and '0' in PCM and tries to solve the power underutilized problem of existing PCM write schemes. The key idea behind Tetris Write is to monitor the number of '1' and '0' changed in each data unit, and schedule the order of data units' write-1 and write-0 execution considering both the time and power asymmetries, to make the best use of power budget. Tetris Write tries to schedule the dominating long term write-1s first and attempts to steal interspaces remained by write-1s to put the extraessential short write-0s. 4-core PARSEC benchmarks' results show that Tetris Write can get 65% read latency reduction, 40% write latency reduction and 46% running time reduction compared with the baseline on average. In addition, Tetris Write can earn 26%, 15% and 10% more read latency reduction, 15%, 7% and 5% more write latency reduction, and outperform 22%, 12% and 7% more running time reduction, compared with the state-of-the-art Flip-N-Write, 2-Stage-Write and Three-Stage-Write schemes, respectively. Moreover, Tetris Write gets 2X IPC improvement compared with the baseline while Flip-N-Write, 2-Stage-Write and Three-Stage-Write earn 1.4X, 1.6X and 1.8X IPC improvements, respectively.

Acknowledgment

This work was supported by the National High Technology Research and Development Program (863 Program) No.2015AA015301, No.2013AA013203, No.2015AA016701; NSFC No.61173043, No.61303046, No.61472153; State Key Laboratory of Computer Architecture, No.CARCH201505; Wuhan Applied Basic Research Project (No.2015010101010004); This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

References

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

[2] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *Proc.ISCA*, pp. 24–33, 2009.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," *Proc.ISCA*, pp. 2–13, 2009.

[4] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *Proc.ISCA*, pp. 14–23, 2009.

[5] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," *Proc.MICRO*, pp. 14–23, 2009.

[6] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *Proc.ISCA*, pp. 383–394, 2010.

[7] X. Luo, D. Liu, K. Zhong *et al.*, "Enhancing lifetime of NVM-based main memory with bit shifting and flipping," *Proc.RTCSA*, pp. 1–7, 2014.

[8] S. Kang, W. Y. Cho, B.-H. Cho *et al.*, "A 0.1-μm 1.8-v 256-mb phase-change random access memory (PRAM) with 66-mhz synchronous burst-read operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 210–218, 2007.

[9] K.-J. Lee, B.-H. Cho, W.-Y. Cho *et al.*, "A 90 nm 1.8 V 512 Mb diode-switch PRAM with 266 MB/s read throughput," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 150–162, 2008.

[10] Z. Li, F. Wang, Y. Hua, W. Tong, J. Liu, Y. Chen, and D. Feng, "Exploiting more parallelism from write operations on PCM," *Proc.DATE*, pp. 768–773, 2016.

[11] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," *Proc.HPCA*, pp. 282–293, 2013.

[12] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing PCM banks from seizing too much power," *Proc.MICRO*, pp. 186–195, 2011.

[13] J. Yue and Y. Zhu, "Making write less blocking for read accesses in phase change memory," *Proc.MASCOTS*, pp. 269–277, 2012.

[14] S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," *Proc.MICRO*, pp. 347–357, 2009.

[15] J. Yue and Y. Zhu, "Exploiting subarrays inside a bank to improve phase change memory performance," *Proc.DATE*, pp. 386–391, 2013.

[16] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "FPB: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," *Proc.MICRO*, pp. 1–12, 2012.

[17] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Bit Mapping for Balanced PCM Cell Programming," *Proc.ISCA*, pp. 428–439, 2013.

[18] F. Xia, D. Jiang, J. Xiong, M. Chen, L. Zhang, and N. Sun, "DWC: Dynamic write consolidation for phase change memory systems," *Proc.ICS*, pp. 211–220, 2014.

[19] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's next-generation server processor," *IEEE micro*, no. 2, pp. 7–15, 2010.

[20] J. Warnock, Y. Chan, W. Huott *et al.*, "A 5.2 GHz microprocessor chip for the IBM zEnterprise system," *Proc.ISSCC*, pp. 70–72, 2011.

[21] Y. Li, X. Li, L. Ju, and Z. Jia, "A three-stage-write scheme with flip-bit for PCM main memory," *Proc.ASP-DAC*, pp. 328–333, 2015.

[22] R. Maddah, S. M. Seyedzadeh, and R. Melhem, "Cafo: Cost aware flip optimization for asymmetric memories," *Proc.HPCA*, pp. 320–330, 2015.

[23] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "PreSET: Improving Performance of Phase Change Memories by Exploiting Asymmetry in Write Times," *Proc.ISCA*, pp. 380–391, 2012.

[24] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Monta, "Improving read performance of phase change memories via write cancellation and write pausing," *Proc.HPCA*, pp. 1–11, 2010.

[25] "Vivado High-Level Synthesis," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[26] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," *Proc.PACT*, pp. 72–81, 2008.

[27] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems," *Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, July 2015.

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

[29] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[30] B.-D. ang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," *Proc.ISCAS*, pp. 3014–3017, 2007.