# Accelerating Write by Exploiting PCM Asymmetries

Jianhui Yue, Yifeng Zhu

*Electrical and Computer Engineering*
*University of Maine*
{*jianhui.yue, yifeng.zhu*}*@maine.edu*

***Abstract*— To improve the write performance of PCM, this paper proposes a new write scheme, called two-stage-write, which leverages the speed and power difference between writing a zero bit and writing a one bit. Writing a one takes longer time but less electrical current than writing a zero. We propose to divide a write into stages: in the write-0 stage all zeros are written at an accelerated speed, and in the write-1 stage stage, all ones are written with increased parallelism, without violating power constraints. We also present a new coding scheme to improve the speed of the write-1 stage by further increasing the number of bits that can be written to PCM in parallel. Based on simulation experiments of a multi-core processor under various SPEC CPU 2006 workloads, our proposed techniques can reduce the memory latency of standard PCM by 68.3% and improve the system performance by 33.9% on average. In addition, the proposed two-stage-write shows 16.5% latency reduction and 9.2% performance improvement over Flip-N-Write.**

## I. Introduction

Exascale computing will require 1000 times more memory than available today [1]. However, today's DRAM technology is hitting the wall of energy efficiency and transistor scaling. It is a great challenge to fabricate high density DRAM beyond 22*nm* [2] due to difficulties such as efficient charge placement and capacitor control, and reliable charge sensing [3]. Energy consumption and heat dissipation of DRAM with large capacities is a severe issue. In current generation technology, DRAM power consumption can reach 40% of the server energy consumption. The idle power consumption of DRAM accounts for more than 40% of the DRAM power usage [4]. In addition, DRAM's leakage power increases with its capacity and can be as much as its dynamic power [5].

Fortunately, phase-change memory (PCM) has better process scalability and less leakage power. Applying varying levels of currents to phase-change material, such as Ge2Sb2Te5(GST), can transform the material into either crystalline or amorphous states that have dramatically different electrical resistances. This resistance-based memory proves to be very scalable. Furthermore, PCM consumes much less leakage current and requires no refresh operations due to its non-volatile nature. These attractive properties make PCM a viable alternative to DRAM in the near future.

PCM has two major weaknesses: slow write performance and weak write endurance. A PCM chip circuit often limits the maximum instantaneous power due to noise minimization, and hence the number of bits that can be concurrently written is limited to a predefined constant $N$. Typical values of $N$ are 2, 4, 8 and 16. This constraint limits the number of bytes that can be simultaneously written to a bank. $N$ is referred to as a *write unit* in this paper. Accordingly, writing a cache line of 64 bytes requires multiple serially executed write units, which slows down the overall write performance dramatically. Beside slow write, a PCM cell endures around $10^8 - 10^{10}$ write cycles while a DRAM cell can support over $10^{15}$ writes. While many researches are working on the write endurance [6]–[11], in this paper we focus on addressing the issue of slow write.

Applying various amount of electrical current to a PCM cell can make the cell have different resistances. This property is used to store digital information in a PCM cell. In Single-Level-Cell(SLC) technology, a cell with high resistance represents zero, and a cell with low resistance represents one. In Multiple-Level-Cell(MLC) technology, a cell stores multiple bits of information by encoding resistance into several levels. In a SLC PCM, the response time and power requirement of writing a zero and writing a one to a cell are drastically different. Writing a zero takes much shorter time but requires much larger electric current, compared to writing a one. Conventional PCM systems are very conservative in scheduling write requests. Regardless of data content to be written, it is assumed that (1) the response time of writing a binary bit to each cell is as much as required to write a one and (2) the electric current of writing a bit to a cell is as large as required to write a zero. When writing a data block with a mix of ones and zeros, the write bandwidth is degraded due to unnecessary blocking caused by the response time difference between writing ones and zeros, and wasted parallelism caused by under-utilization of electric current supply.

We propose a new PCM write scheme, called *two-stage-write*, which leverages the asymmetric properties for writing a one and a zero to speed up the write operations of zeros and to increase the degree of parallelism for writing ones. The key idea is to separate the write process of a cache line into two separated stages: *write-1 stage* that writes all one bits of the target cache line, and *write-0 stage* that writes all zero bits. Since writing a zero is much faster than writing a one, all writes in the write-0 stage can be performed at an accelerated speed. In the write-1 stage, since writing a one takes much less electric current than writing a zero, more PCM cells can be written concurrently to increase the degree of write parallelism without violating any instantaneous power constraints. Compared with conventional writing schemes in PCM, our two-stage-write can achieve better resource utilization and reduce the service time of writing a cache line.

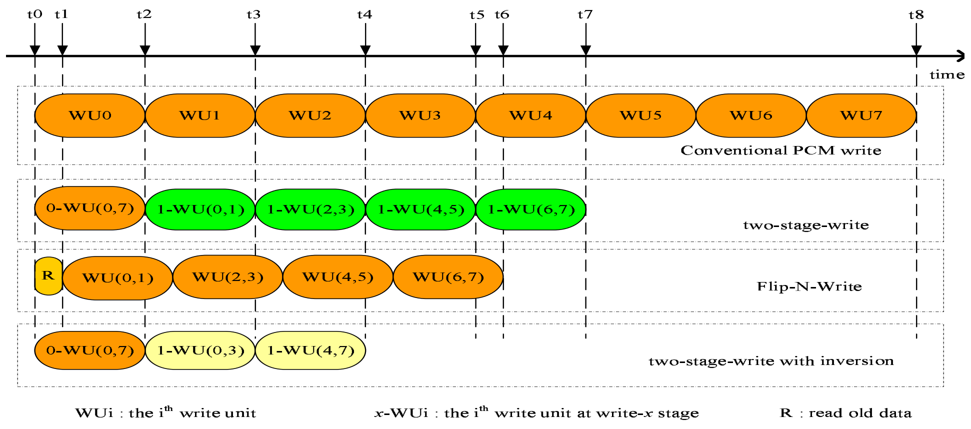


Fig. 1. Timing diagram for different PCM writing schemes

Figure 1 uses a simple example to highlight the difference between the conventional PCM write scheme and our proposed two-stage-write. Assume a cache line has 8 write units, i.e., it requires eight writes to store a cache line. Under the conventional scheme, each write unit completes after a service time required to write a one, regardless of the actual values written. The conventional scheme completes eight write units serially at the time instant $t8$. In two-stage-write all zeros are written to PCM cells at a faster speed and the write-0 stage completes at $t2$. During the following write-1 stage, the lower current requirement of writing a one enables more ones to be written concurrently under the same power budget. Therefore, the number of serial writes in the write-1 stage is reduced. Two-stage-write completes at $t7$.

In two-stage-write, we accurately estimate write current requirements. At each stage, all bits to be written have the same value and thus each bit takes the same amount of electric current. Thus the total current required can be easily calculated. Note that in reality not all bits of a data block to be written are the same. As a result, we only consider bits with a value of $x$ at the write-$x$ stage. In this way, we can have an accurate estimate of electric current requirement for each write.

In addition, we propose a very simple Flip-or-Not-Flip coding scheme to further improve the performance of the write-1 stage. Our goal is to guarantee that the number of bits with the specified value, $k$, in each data block is no larger than half of the size of this data block. Assuming $k$ is 1, this coding scheme examines whether the number of ones is greater than the number of zeros in a given data block. If yes, it flips all bits of this data block. In this way, we ensures that the number of ones does not exceed half of the size of this data block. Under the same electric current constraint, this coding scheme can double the number of ones that can be written concurrently to PCM cells, compared to conventional schemes. Since the write-1 stage dominates the response time of writing a cache line, we apply this coding scheme to the write-1 stage. We call this write scheme two-stage-write-inv. The bit flip is very similar to Flip-N-Write [12], which compares the new data against the old data and writes the flipped new data if the number of different bits is larger than half of the size of the data block. Our inverse scheme differs from Flip-N-Write in that we only need to count the number of ones and the zeros in the new data. We do not need to compare against old data and thus our scheme does not have the overhead of reading old data. As a result, our two-stage-write-inv shows superior performance than Flip-N-Write.

Figure 1 illustrates the difference between Flip-N-Write and two-stage-write-inv. Flip-N-Write completes reading old data at $t1$. After comparing the old data and its flipped data, Flip-N-Write effectively reduces the number of bits to be written by half of the data block size, and thus it allows writing two write units concurrently without violating power budget. Flip-N-Write completes the write of a cache line at $t6$. In two-stage-write-inv, the write-0 stage finishes at $t2$, which is the same as two-stage-write. Through selective inversion, the write-1 stage reduces the number of bits to be written by half, and each time it writes twice the amount of data in the two-stage-write. Accordingly, the write-1 stage in two-stage-write-inv finishes earlier at $t4$.

We evaluate our proposed techniques by simulating a multi-core system under the SPEC CPU 2006 benchmark suite. Experimental results under 13 multi-programmed workloads show that two-stage-write and two-stage-write-inv reduce the read latency of the standard PCM systems by 45.8% and 68.3% on average, and the total running time by 21.9% and 33.9%, respectively. We also compare our design with two state-of-the-art write-latency hidden technologies, including Flip-N-Write and Write Cancellation. Two-stage-write-inv achieves an average of 9.2% performance improvement over Flip-N-Write and 48.1% over Write Cancellation.

The rest of paper is organized as follows. Section II introduces the background of PCM technology. Section III presents our two-stage-write architecture design and Section IV presents two-stage-write-inv scheme. Section V discusses the experimental results. Related work is summarized in Section VI and conclusions are given in Section VII.

## II. Background

### A. Phase Change Memory

Phase Change Memory (PCM) is a one-transistor, one-resistor (1T1R) device shown in Figure 2, while DRAM is a one-transistor, one-capacitor (1T1C) device. PCM exploits the remarkably different properties of phase change material in a memory cell to store digital information. Phase change material, such Ge2Sb2Te5(GST), has two phases: an amorphous phase that has high resistance in the $M\Omega$s range and a crystalline phase that has lower resistance in the $K\Omega$s range. PCM uses the resistance associated with a memory cell to represent binary information. By exploiting the large range of resistance and encoding the resistance into multiple levels, a PCM cell can store multiple bits of information called multiple-level-cell (MLC). On the other hand, a cell storing only a single bit of information is a single-level-cell (SLC). However, due to process variations, it is difficult to accurately apply electric currents to induce the GST to the target resistance, due to narrow spaces between neighbor cells. During writes, various current values are tried until it reaches the target resistance. In general, the larger the number of bits stored in a cell, the greater the number of write attempts. As a result, MLC is slower than SLC. In this paper, we focus on SLC due to its relatively fast write speed.

A PCM cell is read by sensing the current flow from the storage location, which is determined by the resistance of the cell. For PCM, write latency is much larger than read latency since GST needs to be heated up to change its phase during a write, as shown in Figure 2.

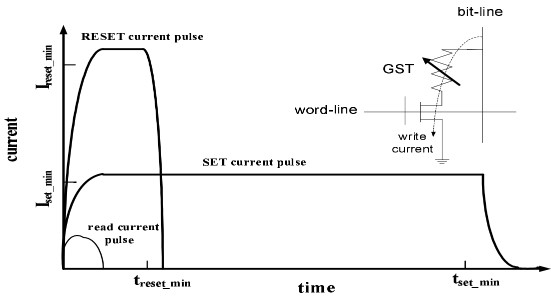


Fig. 2. Electric current required to write a zero and a one to a PCM cell

Writing a zero and writing a one to a PCM cell exhibit remarkable differences in electric current and response time. When a zero is written, a large current is applied to the cell for a short duration in order to heat the GST abruptly and move the GST to the amorphous phase. On the other hand, if a one is written, a relatively smaller current is applied to the cell for a longer duration. The GST is slowly heated and remains in the crystalline phase after reaching a certain temperature. Besides the difference in write currents, writing a one takes a longer time than writing a zero. Writing a zero changes the GST to have high resistance. Large currents induce the GST to high resistance state easily despite the process variations. However, writing a one is more challenging than writing a zero due to process variations. In high density PCM cells, it is difficult to control variations of cells during the fabrication process, which makes it almost impossible to precisely program a PCM cell. Typically a programming-verify (PV) technique is used in industry and academic research. The micro-controller in the PCM heuristically chooses an initial programming pulse for the cell. After programming, the micro-controller reads this cell's resistance and compares it against the target resistance. If the cell has not reached the target resistance, another pulse is tried. This process is repeated until the cell achieves the target resistance. These multiple iterations of verification further increases write latency. This is why writing a one is much slower than writing a zero.

Since zeros and ones are randomly distributed among memory cells, the memory controller chooses the slowest time, i.e., the time to write a one, as the required waiting time after writing a bit. Meanwhile, the conventional PCM also conservatively assumes writing any bit needs the same amount of electric current as required to write a zero.

### B. PCM Division Write Modes

The write performance is also limited by the electronic circuity inside a PCM chip. As discussed previously, writing a zero bit requires a large amount of electric current to heat GST. On a write, the bit line is raised to a voltage higher than the phase change voltage, typically ranging from Vdd+1 to Vdd+3, resulting in a higher power draw. The Dickson charge pump, widely used inside the PCM chip, provides the current to the write driver. The noise on the power line hinders the charge pump from providing the high level of instantaneous current needed for writing PCM cells [13] inside a chip. In addition, the low efficiency of the charge pump further limits parallel writes [14]. This limitation of current provision constrains the number of concurrent write bits to 2, 4, or 8 in a chip [13]. This writing scheme, referred to as write division mode [14], results in increased latency when writing large data. For example, writing 16 bits to a PCM chip takes 8, 4 and 2 time units when writing under x2, x4 and x8 write division modes, respectively. Conventional write schemes conservatively assume that each bit of a data block to be written is a current-consuming bit (i.e., a zero bit) and thus overestimate the electric current requirement to write a data block. This conservative estimate results in a loss of opportunity to write more data in one write operation. In this paper we propose two methods to accurately estimate the write current and improve PCM write performance under the same current delivery capacity as the conventional PCM write scheme.

## III. Two-stage-Write Scheme

### A. Motivations and Design

Writing a zero and writing a one to a PCM cell have distinct response times and electric current requirements. Compared to writing a one, writing a zero takes much larger electric current to the heat the GST but for a much shorter amount of time. The maximum instantaneous electric current that a PCM chip can provide often limits the number of bits that can be concurrently written. When writing a data block with a mix of zeros and ones into PCM cells concurrently, conventional PCM controllers have to wait for the completion of the last bit. Thus the PCM controller can not issue an outstanding request even though all zeros in the current write request have been successfully written. In addition, a conventional PCM controller always prepares for the worst case instantaneous electric current demand and assumes all bits written are zeros. Based on this assumption, the number of bits that can be concurrently written are fixed to a small number. Consequently, the system resources tend to be under-utilized, leading to inferior write performance.

We propose a two-stage-write scheme that fully exploits the asymmetric properties between writing a zero and writing a one to solve the under-utilization issue discussed above. When writing a block of data (typically a cache line) our write scheme takes two steps. In the first step, called *write-0 stage*, all zeros are written to PCM cells at a high speed. In the second step, called *write-1 stage*, all ones are written with an increased write unit size. Like the conventional write scheme, the write-0 stage still complies with the parallel write limit and thus writing a cache line needs multiple serial write operations. However, the write-0 stage is performed at a much faster speed because no ones are written in this stage. The write-1 stage can write more bits in parallel than conventional PCM write schemes. We leverage the fact that writing a one takes less current than writing a zero, and thus we can increase the size of the write unit without violating instantaneous power constraints. With a large write unit size, the number of serial writes required to write a cache line is reduced accordingly.

Our novelty lies in exclusively writing ones and zeros at two separated stages to fully leverage the shorter time required to write a zero and less electric current required to write a one. In conventional write schemes, ones and zeros are mixed in each write request, and thus the controller has to conservatively limit the number of bits that can be concurrently written to the worst case scenario in which all bits are ones, resulting in power supply under-utilization. For example, as shown in Figure 1, the write operations of WU0, WU1, . . ., and WU7 take exactly the same amount of time under the conventional PCM write schemes. Compared with conventional write schemes, our scheme can write all zeros with a faster speed and write all ones with a larger degree of parallelism, under the same power constraints. In this example, the write-0 stage, including 0-WU(0 ˜7), completes at $t_2$ and it performs faster due to shorter time required to write zeros. The write-1 stage, including 1-WU(0˜7), can write two data blocks each time due to lesser power requirement for writing a one, thus significantly reducing the overall time to write data block WU(0 ˜7). In the two-stage-write, ones and zeros are separated and thus the number of concurrently bits written can be precisely determined in each stage. The advantage of two-stage-write is that we can write all zeros with a tighter schedule and all ones with a larger write bandwidth. While the extra write-0 stage generates some performance overhead, the accelerated write-1 stage achieves significant performance gain compared with other write schemes. In essence, the 2-stage write trades the larger write unit size during writing ones for the shorter response time during writing zeros.

## B. Analysis of Write Service Time

We analyze the response time of writing a cache line in our two-stage-write and the conventional write scheme. Assume it takes $T_{reset}$ time to write a zero bit, and $T_{set}$ time to write a one bit. Writing a one is much slower than writing a zero, and we let $T_{set} = K \times T_{reset}$, where $K$ is the time ratio and is set to 8 in Ref. [15]. Assume the electric current required to write a zero is $L$ times of the current to write a one, and $L$ is 2 in Ref. [3]. The write unit size is $M$ bytes and a cache line size has $N$ bytes. For simplicity, we let $M = 8$ and $N = 64$, and assumes zeros and ones exist in a data block with an equal probability. Time to write a cache line under the conventional write scheme and two-stages-write scheme can be calculated by the following two equations.

$$T_{conventional} = \frac{N}{M} K T_{reset} \tag{1}$$

$$\begin{aligned} T_{2stages} &= \frac{N}{M} T_{reset} + \frac{N}{ML} K T_{reset} \\ &= (\frac{1}{K} + \frac{1}{L}) \frac{N}{M} K T_{reset} \\ &= (\frac{1}{K} + \frac{1}{L}) T_{conventional} \end{aligned} \tag{2}$$

As shown in Eqn. 2, the write-0 stage has $N/M$ serial writes and its total response time is $\frac{N}{M}$ $T_{reset}$. The write-1 stage increases the write unit size by a factor of $L$. Typically the write-0 stage takes shorter time than the write-1 stage. Suppose the power ratio $L$ is 2 [3] and the time ratio $K$ is 8 [15], according to Eqn. 1 and 2, we have $T_{conventional} = 64T_{reset}$ and $T_{2stages} = 40T_{reset}$. This simple calculation shows that the time for writing a cache line in two-stage-write is only 62.5% of conventional write.

$$speedup = \frac{T_{conventional}}{T_{2stages}} = \frac{1}{\frac{1}{K} + \frac{1}{L}} \tag{3}$$

Eqn. 3 indicates that the speedup of our two-stage-write scheme over the conventional scheme is not sensitive to the degree of write division mode, instead sensitive to the time ratio $K$ and the power ratio $L$ if the write unit size is smaller than a cache line. In Ref. [15], the largest degree of division write mode is 16 and thus the corresponding write unit is 8 bytes, which is smaller than a cache line (typically 64 bytes). If the write division mode increases, the speedup will not be affected since the speedup is not sensitive to it when the write unit is smaller than a cache line. Even if the write unit increases in the future, we believe that the cache line will also increase to reduce the tag overhead in ever larger cache capacity and the cache line would be still larger than the write unit. For example, the size of the last-level cache line has been increased to 128 bytes in IBM POWER7 [16] and 256 bytes in IBM zEnterprise [17].

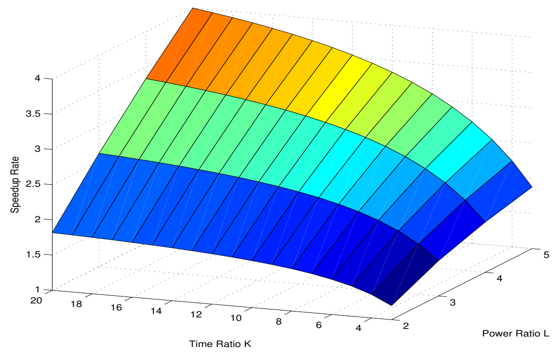


Fig. 3. Speedup of two-stage-write over the baseline

Figure 3 shows the speedup of two-stage-write over the baseline as the time ratio $K$ and the power ratio $L$ vary in their representative range. $K$ is typically between 3.75 to 18. $L$ is typically between 2 and 3. For example, $K$ is 15 in Ref. [18], 3.75 in Ref. [19], 18 in Ref. [20], 6 in Ref. [21], and 8 in Ref. [15]. $L$ is 2 in Ref. [19], 3 in Ref. [20], and 2 in Ref. [21]. As $K$ and $L$ varies under their representative range, we can see that the speedup is constantly larger than one.

$$T_{FNW} = \frac{N}{M} T_{read} + \frac{N}{2M} K T_{reset} \tag{4}$$

We also compare two-stage-write with Flip-N-Write (FNW). In Eqn. 4, FNW needs to read data out first for bit-wise comparison. On average, only half of the bits will be written into PCM cells after comparison. Since $T_{reset}$ is very close to $T_{read}$ [15], we can have $T_{FNW} = 33T_{reset}$, which is slightly better than two-stage-write ($T_{2stages} = 40T_{reset}$). Our experimental results presented later will show that this writing time gap causes 4.3% increase in read latency and 2.6% performance degradation on average in two-stage-write under various 4-core SPEC 2006 CPU workloads. However, our two-stage-write has no overhead of storing data inversion flags used by Flip-N-Write, which needs 6.25% of the total PCM storage capacity in a typical setting.

## C. Implementation

Figure 4 presents our proposed implementation of two-stage-write based on the product-grade PCM prototype from Samsung [15], whose data path is illustrated in Figure 4. This prototype supports eight-word prefetch by introducing a buffer for reading requests and limits the maximum number of parallel written bits to 16 for each chip. The buffer is connected with multiple sense amplifiers and write drivers. Our two-stage-write circuit sits in the data path between the x128 buffer and these write drivers as shown in Figure 4. This circuit can write 16 bits of 0; writing 128 bits needs 8 serial writes in the write-0 stage. Since this PCM prototype does not disclose the current ratio between writing a zero and a one in Ref. [15], we assume this ratio is 2 according to other PCM prototypes [3]. As a result, two-stage-write can write 32 ones each time and it takes 4 serial writes to write 128 bits in the

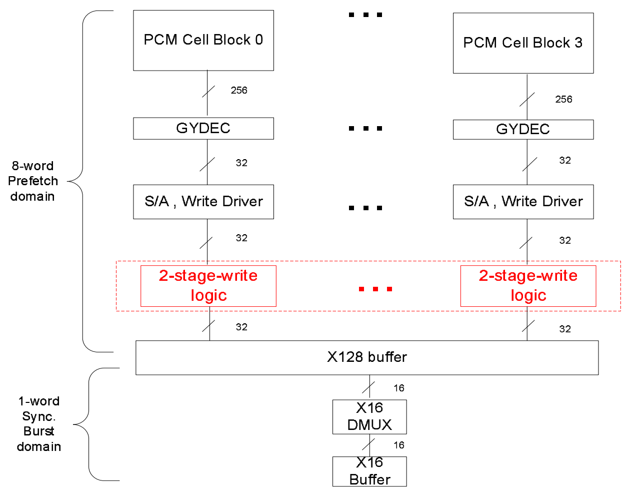


Fig. 4. PCM data path

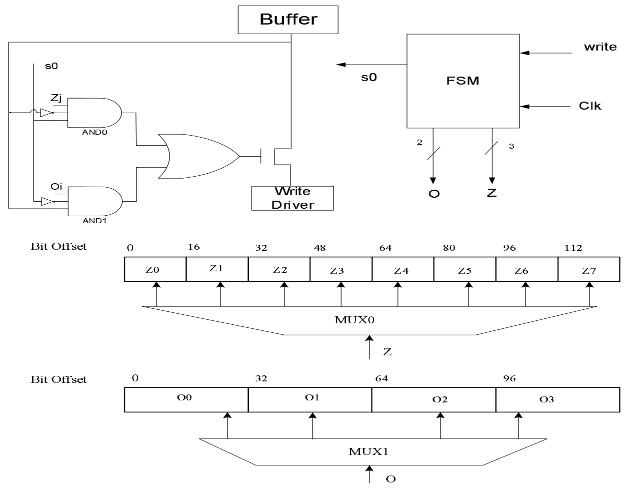


Fig. 5. Two-Stage-Write Control

write-1 stage. Each serial write has a corresponding region of bits in the buffer, which can be specified by the offset. We call this region the write-region.

Our two-stage-write circuit includes individual component for each bit and a common component shared by all bits in the buffer as shown in Figure 5. The shared finite state machine (FSM) produces the output control signals for the write-stage (s0), the specified serial write to write a zero (Z), and the specified part of buffer to write a one (O). If s0 is 1, the write is at the write-0 stage; otherwise, it is at the write-1 stage. Since different write-regions are written at different stages (see Figure 5), the O and Z produced by the FSM specify which write-region is scheduled respectively at the write-1 stage and write-0 stage. In our design, the write-1 stage writes 4 write-regions and write-0 stage writes 8 write-regions. The O signal has 3 bits and the Z signal has 2 bits. Each bit in the buffer is an input of individual component shown in Figure 5. When the buffer stores a zero, the write-0 stage sets s0 to be 1 and signals the corresponding $Z_j$, which means this buffer belongs to the $j^{th}$ serial write in the stage-0, and enables the AND0 gate and passes this zero to the PCM cell. When the buffer stores a one, the write-1 stage set s0 to be 0 and signals the corresponding $O_i$, which means this buffer belongs to the $i^{th}$ serial write in the stage-1, and enables the AND1 gate and passes one to the PCM cell.

The input $Z_j$ and $O_i$ for each bit specify whether this bit is involved in the $j^{th}$ and $i^{th}$ serial write in the write-0 stage and write-1 stage, respectively. For example, the first serial write in the write-0 stage involves 16 bits whose offset ranges from 0 to 15 in the buffer under the x16 write division mode and each bit in the $j^{th}$ write-region in the write-0 stage is associated with its serial writing order $Zj$ as shown in Figure 5. Since each PCM has a 128-bit buffer, two-stage-write has 8 serial writes in the write-0 stage. However, the write-1 stage only has 4 serial writes. This is because the current for writing a one is half of the current of writing a zero, and 32 ones can be concurrently written without exceeding current supply budget under the x16 write division mode. As a result, the first write region in write-1 ranges from 0 to 31 in the buffer as shown in Figure 5. Since Z and O produced by the FSM can indicate the order of the specified write at different write stages, MUX0 and MUX1 use Z and O to specify the corresponding part of data in the buffer for the specified write-0 and write-1 operation respectively. Because the write operation is not in the performance critical path, the overhead associated with two-stage-write circuits is negligible. Note that this implementation only modifies the write data path and keeps the read data path unchanged.

Integrating our 2-stage write logic into PCM chips has little overhead. First of all, the control logic is not in the list of the most cost sensitive components in DRAM [22] and PCM. Second, the added components in our controller are much less complicated than the components that are already widely used. For example, the PCM chips program-and-verification logic is much more complex than our logic, which handles the process variations. In addition, the content-aware writing logic is commonly used in NAND Flash memory. Third, our experiment results show that our 2-stage write has negligible performance overhead, very similar to Flip-N-Write.

We have analyzed the two-stage-write scheme and demonstrated the two-stage-write scheme performance advantage over the conventional scheme. However, this advantage is limited by the write current ratio between writing a zero and writing a one. For example, this ratio is reported to be 2 in Ref. [3]. In the following, we introduce a new coding scheme to further improve the performance of two-stage-write.

## IV. Two-Stage Write with Inversion

In two-stage-write, we make an accurate-yet-conservative estimate of write current requirements. In our approach, each stage assumes all bits of a write unit have the same value and the electric current is estimated to write these bits. Each stage of the two-stage-write only writes a portion of the data block which consists of either zeros or ones. Thus, we could have more accurate estimate of the electric current requirement. However, the distribution of zeros and ones over a block of data changes dynamically, it is difficult to predict how many zeros and ones are actually written at each stage. In order to address this issue, we introduce a simple data coding scheme to guarantee the number of bits with the specified value, $k$, in each data block is not larger than half the size of this data block. Since the write-1 stage dominates the writing for a cache line in the two-stage-write, we set $k = 1$.

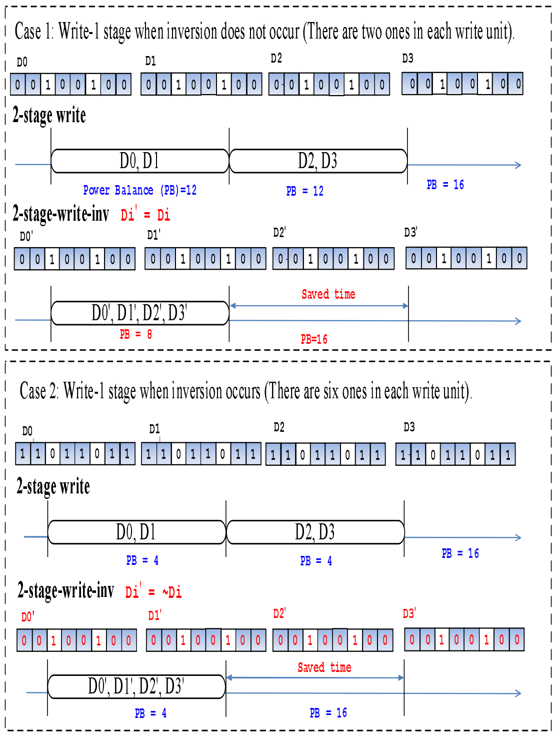
We present our coding scheme in Algorithm 1. Before writing a write unit, we count the number of bits that are one (line 5). If the occurrences of ones are more than half of the write unit size (line 6), all bits in this write unit are inverted (line 7). Otherwise, the write unit is kept unchanged (line 10). In this way, we can guarantee that the number of ones to be written is less than half of the write-1 stage without violating the instantaneous current constraints. Combined with basic two-stage write introduced previously, this selective bit-wise inversion can further improve the write performance. We call this combined write scheme as *two-stage-write-inv*. In addition, the two-stage-write-inv introduces an extra bit, $I$ shown in Alg. 1, to store inversion flag for the corresponding sub-block in order to correctly read its data.

**Algorithm 1** *Data Inversion for two-stage-write*

```
1: /* $N$: write unit size in bits */
2: /* $I$: data inversion flag */
3: /* $D$: data to write */
4: /*count the number of ones in D */
5: $cnt$ = $get\_Num\_Of\_Ones(D)$
6: if $cnt > N/2$ then
7:     $D$ = $\sim D$
8:     $I$ = 1
9: else
10:    $I$ = 0
11: end if
```

We use a simple example, as shown in Figure 6, to illustrate how two-stage-write-inv works in the write-1 stage. Assume a bank has a power budget of 16 and that writing a zero and a one consume 2 and 1 power units respectively. A total of 4 bytes (D0, D1, D2, and D3) are to be written to the PCM. In the first case, assume each byte has 2 ones. Then the two-stage-write with no inversion can write 2 bytes each time even though the bank has enough power balance to write more bits. Since the number of ones in each byte in the first case is less than 4, two-stage-write-inv can write 4 bytes without violating the power budget. Now for the second case, assume each byte has 6 ones. Thus 4 bytes cannot be written concurrently by the two-stage-write without inversion. However, two-stage-write-inv flips the original data first, and then each converted data has only 2 ones. Accordingly, it can safely write 4 bytes to PCM without violating the power constraint. Note that Fig. 6 does not show inversion flag in order to highlight the key idea behind two-stage-write-inv.

$$T_{2stages\_inv} = \frac{N}{M}T_{reset} + \frac{N}{2ML}KT_{reset} \qquad (5)$$



Case 1: Write-1 stage when inversion does not occur (There are two ones in each write unit).
D0
D1
D2
D3
2-stage write
D0, D1
D2, D3
Power Balance (PB)=12
PB = 12
PB = 16
2-stage-write-inv Di' = Di
D0'
D1'
D2'
D3'
D0', D1', D2', D3'
Saved time
PB = 8
PB=16
Case 2: Write-1 stage when inversion occurs (There are six ones in each write unit).
D0
D1
D2
D3
2-stage write
D0, D1
D2, D3
PB = 4
PB = 4
PB = 16
2-stage-write-inv Di' = ~Di
D0'
D1'
D2'
D3'
D0', D1', D2', D3'
Saved time
PB = 4
PB = 16

Fig. 6. Comparing the write-1 stage of two-stage-write inversion with two-stage-write

Two-stage-write-inv takes less time to write a cache line than Flip-N-Write [12], which is one of the state-of-the-art PCM write schemes. Through comparison of old data, Flip-N-Write uses a similar coding scheme to reduce the number of bits to be written by half on average if the overhead is ignored. Our two-stage-write-inv utilizes a different coding scheme and it reduces the number of ones to be written by half on average in the write-1 stage. Utilizing published parameters in Eqn. 4 and Eqn. 5, we can compute the time for writing a cache line under Flip-N-Write and two-stage-write-inv, with an assumption of $T_{read} = T_{reset}$ [15]. Writing a cache line takes $33T_{reset}$ and $24T_{reset}$ respectively for Flip-N-Write and two-stage-write-inv. Such a back-of-the-envelope calculation shows that two-stage-write-inv can be 27% faster than Flip-N-Write.

### A. Implementation

Two-stage-write-inv extends the implementation of two-stage-write by adding an extra circuit to examine and perform bit-wise inversion as shown in Figure 7. Since we selectively invert data based on its number of ones, we use an extra cell to store this inversion flag for a fixed number of data bits and we need to accordingly augment the PCM chip with more bits, as is demonstrated in Ref. [12]. The fixed number of bits is 16 in our design.

The inversion control circuit uses a counter to count the number of ones and its output is an input to a subtractor. The subtractor compares the counter and a constant of 8, which is half of the maximal parallel bits in a chip. The subtractor's

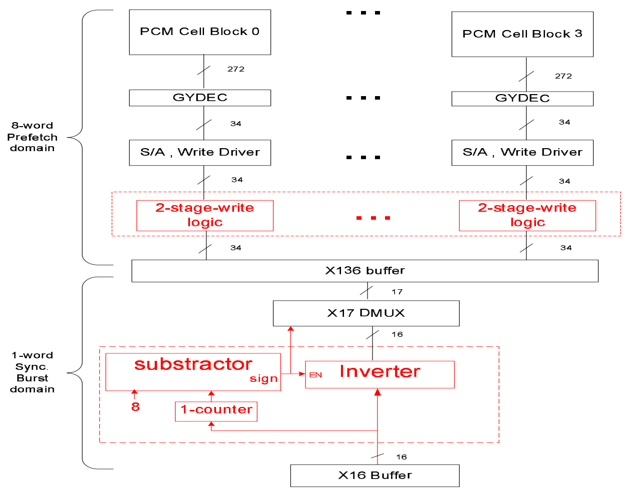


Fig. 7. 2 Stage Write with Inversion

sign is used to control the inversion, and it is also an input to the x17 DMUX to be written to PCM for an inversion flag. Two-stage-write-inv reduces the number of write operations for a cache line, and a FSM in the two-stage-write logic is needed to record the write status. The selective inversion coding algorithm reduces the number of bits to be written by half on average and this reduction translates into a larger write unit size at the write-1 stage. Accordingly we need to make two minor changes to the two-stage-write logic. First, since the number of write units is reduced, the FSM is reduced to the number of write operation in the write-1 stage and its output O. Second, the MUX1 is changed to direct data switching so that two-stage-write-inv can write more bits than two-stage-write as shown in Figure 8.

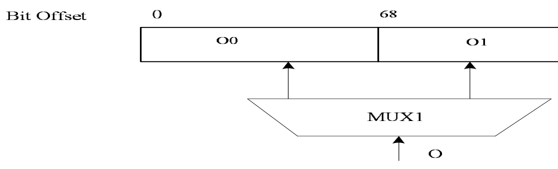


Fig. 8. Modification to MUX1 Logic

Our two-stage-write-inv uses selective data inversion similar to Flip-N-Write to improve PCM write performance. However, our motivations for using data inversion is different. While the Flip-N-Write guarantees the number of modified bits to be written is half of the write unit, our scheme ensures that the number of one bits is less than half of the data size, which is larger than one conventional write unit.

## V. Evaluation

### A. Experimental Methodology

We evaluate the performance by using the execution-driven processor simulator M5 [23] and the cycle-level memory simulator DRAMsim [24], which is modified to simulate PCM. Table I shows the parameters of simulated processor and product grade PCM [15]. In order to accurately model memory

| Parameter | Value |
|---|---|
| System | 4-core CMP, 4 GHz |
| Execution Core | Alpha-like out-of-order processor |
| L1 Cache | 32KB I-cache, 32KB D-cache |
| L2 Cache | Latency 20*ns*, 2MB, 4-way, 64B cache line |
| L3 Cache | Latency 50*ns*, 32MB, 8-way, 64B cache line |
| Memory Controller | RIFF request scheduling algorithm, page level interleaving address mapping |
| Width of data bus | 64 bits |
| Number of Ranks | 2 |
| Number of Banks | 16 |
| Number of Chips per Bank | 4 |
| Width of a PCM Chip | 16 bits |
| Time to write a PCM cell | a zero bit: 430*ns*, a one bit: 50 *ns* |
| Time to read a PCM cell | 53*ns* |
| Ratio of current writing a zero to a one | 2 |
| PCM write unit size | 8 bytes |

TABLE I
SIMULATION PARAMETERS

access, we use out-of-order cores because it produces more parallel and independent memory accesses than in-order cores. After passing through cache, memory accesses go directly to PCM. The simulated processor has four cores with 32 MB L3 cache. The PCM read and write latency are set to $57ns$ and $430ns$ respectively [15].

We take PCM chip constrains into account, and model the PCM write unit to be 8 bytes. As a result, in conventional writing scheme, writing a cache line of 64 bytes to PCM takes $8 \times 430ns = 3440ns$ while reading a cache line needs less than $100ns$ due to the chip-level prefetch. To tolerate slow PCM write, we add a large off-chip L3 DRAM cache. In addition, since the memory write back is not in the performance critical path, we use the Read Instruction and Fetch First (RIFF) scheduling algorithm to improve the performance [25].

The SPEC CPU 2006 benchmark suite is used to construct 13 multi-programmed workloads with intensive memory accesses. All test applications in each workload run in parallel and each application is fast-forwarded 15 billion instructions and 1 billion instructions are simulated. Table II summarizes memory Read Per Kilo Instructions (RPKI), memory Write Per Kilo Instructions (WPKI), Memory Level Parallelism (MLP) [26] and Bank Level Parallelism (BLP) [27] for each workload. RPKI and WPKI are indicators of memory access intensity of a workload and most of them are larger than 1. While our L3 cache has 32MB and is smaller than the one used in other studies [25], [28], the intensity of memory accesses measured in our workloads is very close to theirs and thus we believe a larger L3 cache is unnecessary.

We compare our design with the baseline scheme that uses the same parameters as listed in Table I but does not have any PCM optimization. We also compare our designs against two recently proposed PCM write optimization techniques including Write Cancellation (WC) [25] and Flip-N-Write (FNW) [12]. Flip-N-Write flips all bits if necessary to reduce by half on average the number of bits actually written to the PCM by comparing against the old data. Write cancellation aborts an on-going write for a newly-arriving read request targeted to the same bank if the write operation is not close to completion. When there are no read requests, all cancelled write requests will be re-executed. The static threshold value for write cancellation is set to be 75%, which was reported to be optimal under SPEC 2006 [25]. We also compare these algorithms against an ideal PCM whose write latency is very small (57*ns*, the same as read latency), which acts as an theoretical upper bound for studying different write optimization techniques for PCM.

We do not compare our designs with write pause [25] and write truncation [28] used in MLC PCM since we focus only on SLC PCM in this paper.

## B. *Two-Stage-Write*

In the PCM simulated in this paper, the electric current to write a zero is twice the current to write a one. Under the same instantaneous power limitation, the number of ones that can be written in parallel is twice the number of zeros. Accordingly, we can double the size of a write unit in the write-1 stage, and four extra write units are needed to write a cache line of 64 bytes.

The time to write a zero is only 1/8 of the time to write a one. On average, the write-1 and write-0 stage take $4 \times 430 = 1720ns$ and $8 \times 50 = 400ns$ respectively to write a cache line under 2-stage write. As a result, it takes a total of $430 + 1720 = 2120ns$ in two-stage-write. Table III summarizes the time required to write a cache line under different write schemes.

*1) Read Latency of Two-Stage-Write :* Figure 9 presents the read latency reduction of two-stage-write and the other schemes, compared with the PCM baseline that does not have any write optimization. In most of the 13 workloads studied, two-stage-write can successfully reduce the read latency more than Write Cancellation (WC). On average, the read latency reduction of two-stage-write is 45.8% less than the baseline, while Write Cancellation and Flip-N-Write(FNW) achieve only 2.89% and 52.1% respectively.

Two-stage-write achieves a lower read latency than Write Cancellation in 10 of 13 workloads. In Write Cancellation, an on-going write still blocks read requests if it has finished more than $x\%$ of the time while writing a cache line, where $x\%$ is a predefined threshold. In our experiments, with a threshold of 75%, Write Cancellation can block a read request for up to 860*ns*. The possibility of blocking a read increases as more re-executions of canceled writes occur. This increases the possibility of read blocking for Write Cancellation and results in larger read latency than two-stage-write, which has no write re-executions. However, it is also noted that Write Cancellation provides the lowest read latency in the workload MIX1. This is because Write Cancellation works well if the bank level parallelism (BLP) of a workload is close to its memory level parallelism (MLP). For example, as shown in Table II, BLP and MLP are 6.02 and 8.11 respectively in MIX1. In workloads where requests are evenly distributed over

| Benchmark | Description | RPKI | WPKI | MLP | BLP |
|---|---|---|---|---|---|
| MIX1 | astar, astar, astar, astar | 7.56 | 4.73 | 8.11 | 6.02 |
| MIX2 | astar, cactusADM, libquantum, soplex | 7.68 | 3.01 | 19.77 | 3.79 |
| MIX3 | cactusADM, cactusADM, gobmk, gobmk | 3.80 | 3.58 | 71.49 | 2.85 |
| MIX4 | gobmk, gobmk, cactusADM, hmmer | 0.97 | 0.58 | 22.99 | 1.45 |
| MIX5 | gobmk, leslie3d, mcf, libquantum | 1.49 | 1.45 | 62.36 | 2.91 |
| MIX6 | gobmk, zeusmp, mcf, lbm | 4.68 | 2.48 | 64.53 | 4.294 |
| MIX7 | leslie3d, bzip2, mcf, lbm | 2.08 | 1.27 | 48.06 | 4.24 |
| MIX8 | leslie3d, gobmk, lbm, astar | 7.07 | 5.08 | 64.62 | 5.62 |
| MIX9 | leslie3d, leslie3d, soplex, soplex | 7.85 | 4.58 | 54.69 | 5.15 |
| MIX10 | leslie3d, soplex, bzip2, astar | 2.10 | 1.29 | 28.19 | 5.22 |
| MIX11 | milc, libquantum, lbm, GemsFDTD | 2.68 | 2.46 | 34.29 | 4.35 |
| MIX12 | soplex, soplex, sjeng, sjeng | 1.61 | 1.01 | 31.70 | 4.92 |
| MIX13 | soplex, soplex, soplex, soplex | 39.11 | 20.70 | 50.18 | 5.45 |

TABLE II
CHARACTERISTICS OF 13 FOUR-CORE WORKLOADS

| Conventional | Flip-N-Write | Two-stage-write | Two-stage-write-inv |
|---|---|---|---|
| 3340*ns* | 1773 *ns* | 2120 *ns* | 1260*ns* |

TABLE III
SERVICE TIME FOR WRITING A CACHE LINE

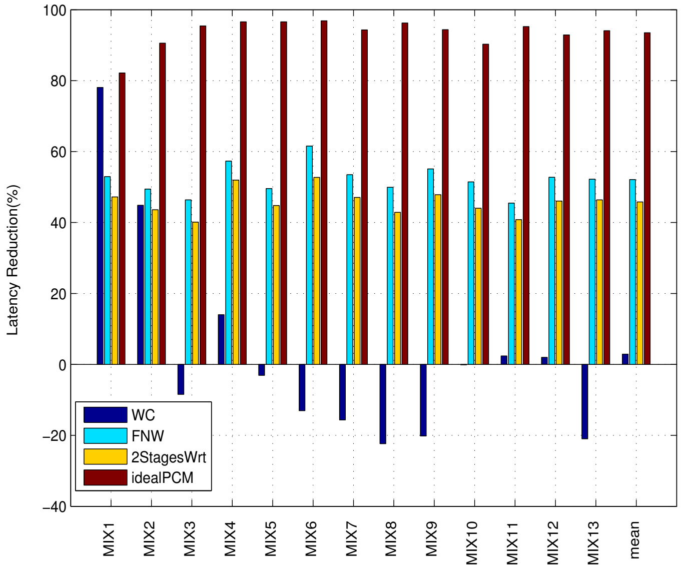


Fig. 9. Two-stage-write Read Latency Reduction

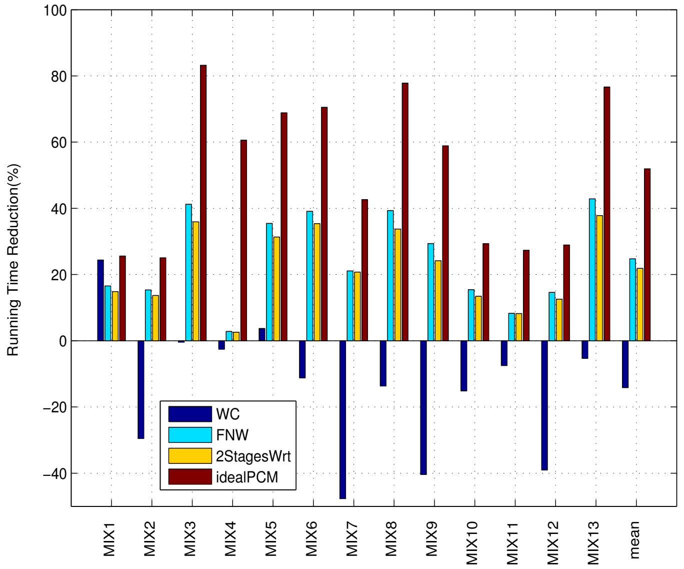


Fig. 10. Two-stage-write Running Time Reduction

different banks, Write Cancellation can reduce the possibility of re-executing canceled writes and hence effectively reduce read latency. We call these workloads Write Cancellation friendly workloads.

However, Write Cancellation is less effective for unfriendly workloads with remarkably different BLP and MLP. As a result, in these unfriendly workloads, most requests are clustered to a smaller number of banks, resulting in high frequent occurrences of write cancellations. Most workloads in our experiments are Write Cancellation unfriendly and accordingly two-stage-write outperforms Write Cancellation in terms of average read latency.

Two-stage-write read latency is close to Flip-N-Write. However, two-stage-write has no storage overhead. With flipping bits, Flip-N-Write doubles the size of a write unit and reduces read latency. Flip-N-Write's performance overhead is reading the old version of data from PCM and this overhead is smaller than two-stage-write's performance overhead because in two-stage-write writing zeros is executed. So the write service time for two-stage-write is 347*ns* longer than Flip-N-Write shown in Table III. This is 10.39% inferior to Flip-N-Write and leads to a 6.3% read latency increase on average. This is because the introduction of large L3 DRAM cache can mitigate the PCM slow write's negative impact on performance. However, Flip-N-Write's needs 6.25% of the PCM storage space to store flipping for bits for data blocks, unlike the two-stage-write.

*2) Running Time of Two-Stage-Write:* Figure 10 shows the total running time reduction of two-stage-write and other schemes when compared with the PCM baseline. Two-stage-write's reduced read latency is directly translated into performance gain. On average, two-stage-write provides 21.9% performance improvement over the baseline. The average running time of two-stage-write is 2.6% longer than Flip-N-Write due to its small overhead of reading old data. Note that under some workloads Write Cancellation increases the total running time but has a smaller read latency than the baseline. For example in the workload MIX2, its read latency is reduced by 44.9% whereas its running time is increased by 22.5% (see Figure 9). This is because the re-execution of canceled writes increases the possibility of blocking read request when an executing write finishes more than 75%. On the other hand, by canceling write requests, Write Cancellation can improve BLP. For example, the BLP of MIX5 is 3.15 and 2.91 respectively in Write Cancellation and the baseline. The performance of MIX5 is influenced by both read latency and BLP. So this explains why MIX5's running time is reduced by 3.72% but read latency is increased by 3.09% when compared with the baseline.

### *C. Two-Stage-Write with Inversion*

*1) Read Latency of Two-Stage-Write with Inversion:* Figure 11 compares the read latency reduction against the baseline for five schemes, including two-stage-write-inv(2StageWrtInv), two-stage-write(2stageWrt), Write Cancellation (WC), Flip-N-Write (FNW), and the ideal PCM case. On average, the latency reduction for two-stage-write-inv is 68.4%, while Flip-N-Write and two-stage-write is 52.1% and 45.8% respectively. Compared with two-stage-write, the read latency of two-stage-write-inv is 22.6% less. This is because it can double the size of write unit in write-1 stage and spend half the time in the write-1 stage, which has dominated in the service time of writing a cache line. Overall, two-stage-write-inv also outperforms Flip-N-Write's read latency by 16.3% on average. Since the write-1 stage dominates the write service time, four folds reduction of this part can certainly help to reduce read latency more than Flip-N-Write in which the speed-up ratio is limited to 2. In summary, the two-stage-write-inv consistently outperform Flip-N-Write and the basic two-stage-write for all workloads in term of read latency.

*2) Running Time of Two-Stage-Write with Inversion:* The running time reduction against the baseline is shown in Figure 12. Since the two-stage-write-inv has a tighter write current allocation than the other writing schemes due to our new coding scheme, it fully utilizes the power supply without violations, and thus can significantly reduce the time of writing a cache line. Under the 13 workloads studied in this paper, two-stage-write-inv achieves 33.9% running time reduction against the baseline on average, and outperforms two-stage-write and Flip-N-Write by 12.9% and 9.2% respectively.

## VI. Related Work

Due to the advantage of scalability, PCM has emerged as a promising non-volatile memory technology which can potentially replace DRAM. Most of the existing research work focuses on solving issues of its write endurance and slow writes in order to make it practical in real systems. Write endurance has received extensive attentions recently. Ref. [6] presents removing redundant data bits, row shifting, and segment swapping to prolong the PCM lifetime. Ref. [7] shows a start-gap wear leveling technique to improve the PCM endurance with negligible overhead. Ref. [8] proposes dynamically replicating memory write data to different pages with disjoint failures and reading data from both pages in case of data corruption based on the observation that it is easy to find two pages with the same failure distribution over storage space. Ref. [9] designs Error-Correcting Pointers (ECP) that permanently encode the locations of failed bits into a table and replaces failed bits with healthy ones in order to correct corrupted bits. Ref. [10] proposes partitioning memory into partitions with at most one failed bit and then uses error correction codes to correct for each partition. Ref. [11] exploits the healthy bits in a faulty block to store remapping information without any storage overhead in PCM and extends the PCM lifetime to over 7 years. These achievements have led to the PCM being more reliable as main memory.

Several research projects have aimed to hide the long write latency of PCM. Ref. [3] adds a buffer to each PCM bank and exploits the data locality to mitigate the slow write. In addition, Ref. [3] further proposes a partial write strategy for a cache line to reduce the amount of data written to PCM. Flip-N-Write [12] is a simple read-modify-write technique to write either flipped or unflipped data to reduce write time. During a write, Flip-N-Write first reads the old value out of PCM and then calculates the number of different bits between the new data and the old data, as well as the number of different bits between the bit-flipped new data and old data, and finally chooses to write either the flipped or unflipped new data depending on which has more unmodified data. This scheme requires an extra bit to record whether the associated data has been flipped or not.

Recently proposed PreSet [29] relates to our work closely. Although we share the same motivation to exploit the PCM writing asymmetry to improve PCM performance, our work significantly differs from PreSet. First, our work not only exploits the asymmetry in time but also asymmetry in power. Taking advantage of the difference in power needs between writing a zero bit and a one bit, we can reduce the time of writing a cache line by increasing the write unit size. Second, while our work does not increase the write traffic compared with the baseline, PreSet can potentially increase the write traffic to the PCM significantly. PreSet issues the PreSet write requests when a dirty cache line arrives at the last level cache and then issues another write request when this dirty cache line is evicted to the PCM. These PreSet requests can generate large extra memory traffic and potentially degrades

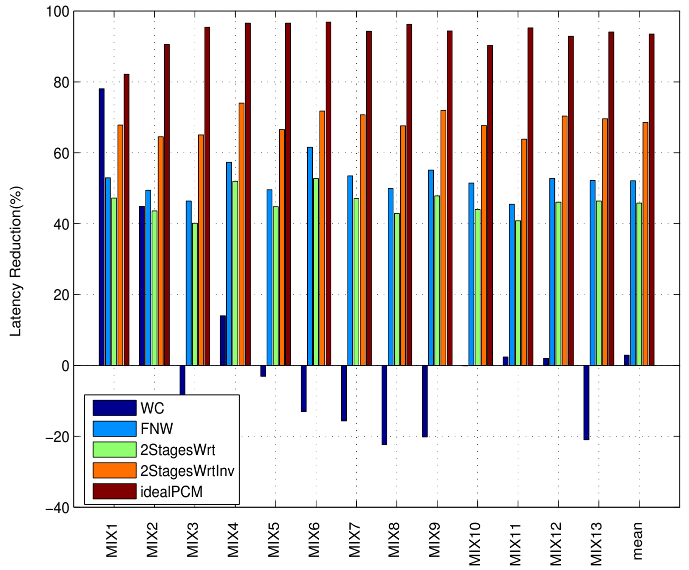


Fig. 11. Read Latency Reduction of Two-Stage-Write with Inversion

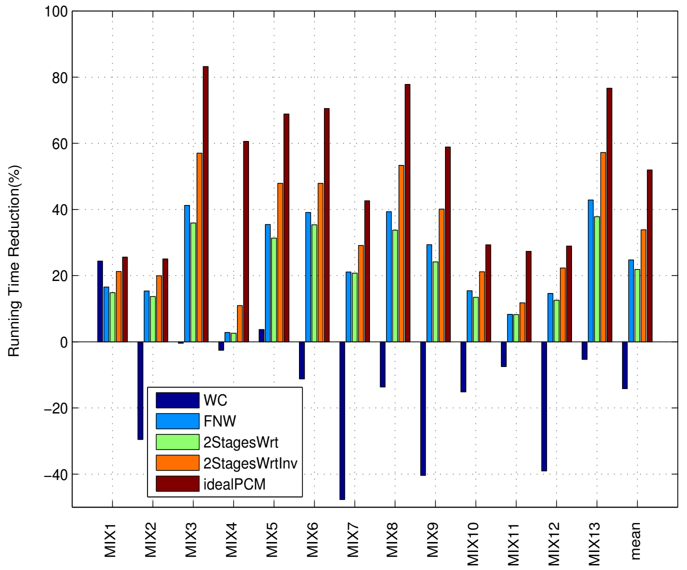


Fig. 12. Running Time Reduction of Two-Stage-Write with Inversion

the system performance under busy memory workloads. However, our method does not incur extra write traffic and can work well under heavy memory workload. Third, the PreSet implementation needs to modify both processors and PCM, our work only needs to make modification to PCM. As a result, our method is easier to deploy. Last, PreSet needs more communication between the processor and PCM than the baseline, our work has exactly the same communication as the baseline. Accordingly, PreSet generates more traffic to on-chip network with a potential negative impact to the overall system performance.

Write cancellation and write pausing [25] are proposed to indirectly improve the PCM read performance. Write cancellation aborts an on-going write for a newly-arriving read request targeted to the same bank if the write operation is not close to completion. When there are no read requests, the cancelled write requests are re-executed. Write pausing is a similar technique that pauses a PCM write at the end of a PCM write iteration and starts to serve a waiting read request. Recently, write truncation and form switch [28] were proposed to improve write performance for the multiple-level-cell PCM. Based on the observation that not all bits for a block of data need the same number of write iterations, the write truncation early terminates the write iteration when most bits have been successfully written and then recovers the data with extra error correction code during reading. The form switch compresses data to reduce the storage space overhead of write truncation.

Besides the reliability and slow write, the PCM needs high levels of electrical current to write to a PCM cell and thermally change its state. Delivering such high levels of power is a challenge for both the PCM chip and the system. A given power budget supply limits the number of parallel write bits in PCM. Ref. [30] proposes power tokens to manage the PCM writes and specifies tight power allocation to increase the number of parallel write bits under a given power budget. Our proposal also estimates minimal power requirement for writing a block and this estimate can efficiently write more bits concurrently. While Ref. [30] minimizes the write power requirement by tracking the number of modified bits in a block, our two-stage-write takes into account the difference between response times and power consumptions for writing a zero and a one. Our two-stage-write proposal results in efficient resource usage for data write to PCM memory. Furthermore, since power token and our methods work at the memory controller level and PCM level respectively, they can work together to improve performance.

## VII. Conclusion

This paper presents and evaluates two new optimization techniques, the two-stage-write and two-stage-write-inv, to better utilize hardware resource and speed up the write performance of PCM. We aim to reduce the negative impact of slow writes on time-critical reads in PCM. Motivated by the PCM's asymmetric properties of response time and electric current when writing a zero and writing a one, we proposes a two-stage-write strategy that writes all zeros first and then writes all ones. Writing all zeros together allows the PCM controller to schedule outstanding requests earlier and writing all ones together allows more bits to be written concurrently without violating instantaneous power constraints.

We propose a simple bit-inverse scheme to further improve the performance of two-stage-writes by doubling the number of ones that can be written in parallel. Specifically when the number of ones in a data block exceeds half of the total number of bits in the data block, then all bits are flipped before performing two-stage-write.

By using the SPEC CPU 2006 benchmark suite to evaluate a four-core system with PCM memory, our experiment results based on 13 different multi-programmed workloads show that two-stage-write and two-stage-write-inv can successfully reduce the read latency by 45.8% and 68.3% on average over a standard PCM baseline, respectively. The read latency reduction directly translates to a running time reduction of 21.9% and 33.9%, respectively. In addition, two-stage-write achieves a performance close to Flip-N-Write, but does not have its storage overhead, which is 6.25% of the capacity of the PCM for Flip-N-Write. Lastly, two-stage-write-inv is superior to Flip-N-Write by 16.5% in latency reduction and 9.2% in running time reduction.

### *A. Acknowledgement*

We would like thank our shepherd Daniel Sorin, anonymous reviewers, and the program committee. We also thank Ali Shareef and Vincent Weaver for their proofreading. This work was supported by National Science Foundation under grants IIS 091663, CNS 1117032, EAR 1027809, CCF 0937988, CCF 0621493, and EPS 0904155.

## References

[1] K. Bergman, S. Borkar, D. Campbell, and etc, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead," 2008.

[2] International Technology Roadmap for Semiconductors, 2009.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 2–13.

[4] H. Zheng and Z. Zhu, "Power and Performance Trade-Offs in Contemporary DRAM System Designs for Multicore Processors," *Computers, IEEE Transactions on*, vol. 59, no. 8, pp. 1033 –1046, Aug. 2010.

[5] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 2008, pp. 51–62.

[6] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 14–23.

[7] M. 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," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 14 –23.

[8] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically Replicated Memory: Building Reliable Systems from Nanoscale Resistive Memories," in *Proceedings of the fifteenth edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, 2010, pp. 3–14.

[9] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for Hard Failures in Resistive Memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 141–152.

[10] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-At-Fault Error Recovery for Memories," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 115–124.

[11] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting Non-Volatile Memory against both Hard and Soft Errors," in *Proceedings of the 17th International Conference on High-Performance Computer Architecture*, 2011, pp. 466–477.

[12] S. Cho and H. Lee, "Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 347–357.

[13] S. Kang, W. Y. Cho, and etc, "A 0.1-$\mu$m 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) With 66-MHz Synchronous Burst-Read Operation," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 1, pp. 210 –218, Jan. 2007.

[14] S. Hanzawa, N. Kitai, and etc, "A 512kB Embedded Phase Change Memory with 416kB/s Write Throughput at 100$\mu$A Cell Write Current," in *IEEE International Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers.*, 2007, pp. 474 –616.

[15] K.-J. Lee, B.-H. Cho, and etc, "A 90 nm 1.8 V 512 Mb Diode-Switch PRAM With 266 MB/s Read Throughput," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 150 –162, Jan. 2008.

[16] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, no. 2, pp. 7–15, Mar. 2010.

[17] J. Warnock, Y. Chan, W. Huott, and etc, "A 5.2GHz Microprocessor Chip for the IBM zEnterprise$^{\mathrm{TM}}$System," in *IEEE International Solid-State Circuits Conference, 2011. ISSCC 2011. Digest of Technical Papers.*, 2011, pp. 70–72.

[18] S. Ahn, Y. Song, and etc, "Highly Manufacturable High Density Phase Change Memory of 64Mb and Beyond," in *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International*, Dec. 2004, pp. 907 – 910.

[19] F. Bedeschi, C. Resta, O. Khouri, and etc, "An 8Mb Demonstrator for High-Density 1.8V Phase-Change Memories," in *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, June 2004, pp. 442 – 445.

[20] H.-R. Oh, B.-H. Cho, W. Y. Cho, and etc, "Enhanced Write Performance of a 64-Mb Phase-Change Random Access Memory," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 1, pp. 122 – 126, Jan. 2006.

[21] S. Kang, W. Cho, K.-J. Lee, and etc, "A 0.1$\mu$m 1.8V 256Mb 66MHz Synchronous Burst PRAM," in *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, Feb. 2006, pp. 487 –496.

[22] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 363 –374.

[23] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[24] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: A Memory System Simulator," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 100–107, 2005.

[25] M. F. Moinuddin K. Qureshi and L. Lastras, " Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing," in *Proceedings of the 16th International Conference on High-Performance Computer Architecture*, 2010, pp. 1–11.

[26] A. Glew., "MLP yes! ILP no! In Wild and Crazy Ideas Session," in *Proceeding of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.

[27] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 2008, pp. 63–74.

[28] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. Childers, "Improving Write Operations in MLC Phase Change Memory," in *Proceedings of the 18th International Conference on High-Performance Computer Architecture*, 2012, pp. 201–210.

[29] M. Qureshi, M. Franceschini, A. Jagmohan, and L. Lastras, "PreSET: Improving Performance of Phase Change Memories by Exploiting Asymmetry in Write Times," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012, pp. 380 –391.

[30] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing PCM Banks from Seizing Too Much Power," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 186–195.