

Improving Write Operations in MLC Phase Change Memory

Lei Jiang †, Bo Zhao†, Youtao Zhang ‡ Jun Yang † Bruce R. Childers‡

† Electrical and Computer Engineering Department ‡ Computer Science Department
University of Pittsburgh University of Pittsburgh
†{lej16,boz6,ju9}@pitt.edu ‡{zhangyt,childers}@cs.pitt.edu

Abstract

Phase change memory (PCM) recently has emerged as a promising technology to meet the fast growing demand for large capacity memory in modern computer systems. In particular, multi-level cell (MLC) PCM that stores multiple bits in a single cell, offers high density with low per-byte fabrication cost. However, despite many advantages, such as good scalability and low leakage, PCM suffers from exceptionally slow write operations, which makes it challenging to be integrated in the memory hierarchy.

In this paper, we propose architectural innovations to improve the access time of MLC PCM. Due to cell process variation, composition fluctuation and the relatively small differences among resistance levels, MLC PCM typically employs an iterative write scheme to achieve precise control, which suffers from large write access latency. To address this issue, we propose write truncation (WT) to reduce the number of write iterations with the assistance of an extra error correction code (ECC). We also propose form switch (FS) to reduce the storage overhead of the ECC. By storing highly compressible lines in SLC form, FS improves read latency as well. Our experimental results show that WT and FS improve the effective write/read latency by 57%/28% respectively, and achieve 26% performance improvement over the state of the art.

1 Introduction

Modern computer systems are increasingly built on chip-multiprocessors (CMPs). With the continuation of Moore's Law, the number of cores in a CMP is projected to increase from today's 4 to 10 cores to dozens, and perhaps even hundreds, of cores in the near future [1]. The scale of memory will grow even larger as the number of cores increases and applications become more data intensive. Unfortunately, this trend jeopardizes current DRAM main memory design. A large capacity DRAM faces power, leakage, and process

variation problems at sub-micron scales. As an example, up to 40% of system power is consumed by main memory in a mid-range IBM eServer [2]. A more severe drawback for DRAM is its scalability. The recent ITRS report [3] indicates there is no path forward to scale DRAM below 22nm.

Phase change memory (PCM) recently emerged as a promising technology to address this looming crisis. PCM has much better scalability [4] than DRAM — a PCM prototype with a feature size as small as 3nm has been fabricated [5]. By adjusting pulse height (i.e., voltage) and width (i.e., duration), it is possible to exploit partial crystallization states to record two or more bits of information in a single cell, which is called multi-level cell (MLC) PCM.

It is challenging to integrate MLC PCM in the memory hierarchy due to its longer access latencies than SLC PCM. Qureshi *et al.* modeled MLC PCM access, and proposed *write cancellation* (WC) [6] and *write pausing* (WP) [6] to let read operations preempt long latency write operations. These techniques reduce the effective read latency of MLC PCM. Qureshi *et al.* further proposed a *morphable memory system* [7] to improve the read latency of MLC PCM by converting one MLC PCM page into two SLC pages when there is sufficient memory. Sun *et al.* [8] proposed to compress data in MLCs for endurance and energy benefits.

MLC PCM usually adopts iterative program-and-verify (P&V) to achieve precise resistance control [4, 7, 9]. However, we observed that most cells can be reliably written in a modest number of iterations, there are typically a small number of cells that require significantly more iterations. The cell that requires the largest number of iterations determines the completion time of one write operation.

Due to the non-deterministic material crystalline process, the same cell may require different number of iterations to complete from write to write. For example, given the same PCM line, the 1st cell may require the largest number of iterations in one write while the 3rd cell is the slowest in the next write. This non-determinism makes it impossible to adopt static designs such as finding and precluding these

cells before the write operation. To address the challenge, we make the following contributions:

- We propose *Write Truncation* (WT) to dynamically identify the cells that require more iterations to write, and truncate their last several iterations to finish a PCM write early. We introduce an extra error correction code (ECC) to cover the erroneous states of those cells. Through truncation, WT significantly reduces the number of iterations of a write operation.
- To mitigate the storage overhead of ECC, we propose *Form Switch* (FS) which uses frequent pattern compression [10] to compress a line to create storage space. If a PCM line can be compressed to less than half of its size, it can be stored in SLC form rather than two-bit MLC form. Since SLC PCM has shorter access latency and better write endurance than MLC PCM, accessing the line as SLC form accelerates performance critical read operations.
- We evaluate our designs in a hybrid DRAM-PCM main memory architecture, and compare them to state-of-the-art designs, such as write pausing [6] and MLC compression [8]. The results show that our designs reduce the effective write and read latencies by 57% and 28% respectively, and achieve 26% performance improvement over the state of the art.

The rest of the paper is organized as follows. Section 2 introduces background and models for PCM and MLC read/write operations. Section 3 presents architectural designs for WT and FS. Section 4 describes our experimental setting and Section 5 discusses our results. Section 6 presents related work and Section 7 concludes our paper.

2 Background

Phase change memory (PCM) is a type of non-volatile memory that stores information in a phase change material such as a chalcogenide alloy. A PCM cell usually consists of a thin layer of chalcogenide with two electrodes on each side. The chalcogenide, such as $\text{Ge}_2\text{Sb}_2\text{Te}_5$ (GST), has stable crystalline (logic “1”) and amorphous states (logic “0”). Given the large resistance contrast between crystalline and amorphous states, it is possible to exploit partial crystallization states to store more than two bits per cell. This approach is referred to as multi-level cell (MLC) PCM. MLC PCM uses smaller resistance ranges to represent different digital levels, which demands more precise control to achieve reliable memory access.

2.1 MLC PCM Write

Writing MLC PCM cells exhibits significant non-determinism due to process variations and composition fluctuation in nano-scale devices. As shown in Figure 1(a), the

composition of Ge, Sb, and Te in GST PCM has both inter- and intra- cell material fluctuations [11]. To change a cell’s resistance level, a large RESET pulse is applied to form an amorphous cap in the GST. After this step, SET pulses are applied to build the crystalline filaments in the cap (Figure 1(b) and Figure 1(c)). The locations of filaments in GST are non-deterministic although the same amount of crystalline volume can be used to represent the same resistance levels. Studies have shown that the filament with slightly more Sb needs less crystallization time [12]. Therefore, given the same SET pulse, different PCM cells, or the same cell at different times, form different filaments in the cap. In addition, the heater size and peripheral circuits also introduce variations to a PCM write [13, 14]. Due to these reasons, it is very difficult for a single current pulse to precisely program a MLC to an intermediate state that has small resistance difference from neighboring states.

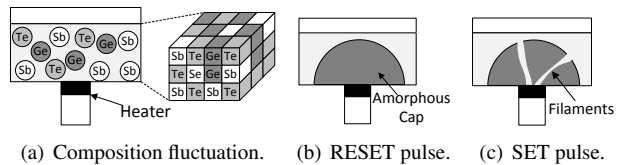


Figure 1. Non-deterministic PCM writes.

Because a single pulse is impractical, an iterative program-and-verify (P&V) strategy is widely adopted in both industrial prototypes [4, 15, 16] and academic research [9]. Given a target resistance range, the write circuit heuristically determines the initial current parameters to program the cell. After the programming pulse, the circuit reads the cell resistance and verifies whether the resistance falls in the target range. The write process completes when the target range is reached. Otherwise, the circuit recalculates the write parameters and repeats the steps until the target range is reached or a predetermined maximum number of iterations has been attempted without success.

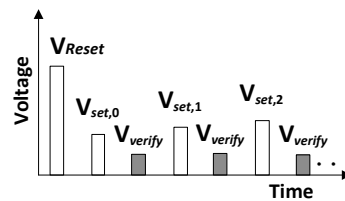


Figure 2. Single RESET multiple SETs staircase-up P&V scheme.

Figure 2 illustrates a staircase-up P&V MLC programming scheme. This design simplifies the hardware by using the same width for all current pulses, starting from an initial *RESET* pulse. An alternative programming scheme uses one SET pulse and multiple RESET pulses [17]. This scheme introduces reliability concerns, including resistance drift. It is also difficult to control the melting process. Therefore,

industrial prototypes and most studies adopt the scheme in Figure 2. We also use the same mechanism, and assume that no write is performed if the value does not change, a.k.a., *Differential Write* [18].

The number of iterations required to complete a particular MLC PCM write depends on many factors, such as required programming precision, programming algorithm, and control circuit design. Qureshi *et al.* developed a two-phase model [6] to capture the P&V PCM programming strategy assuming the convergence probability of write iterations follows the *Bernoulli distribution*. The model considers both the learning phase (the first several write iterations) and the practice phase, and computes the probability to finish a MLC PCM cell write at iteration k as

$$P(k) = \begin{cases} F_1 \cdot (1 - F_1)^{k-1} & \text{if } k \leq i \\ F_2 \cdot (1 - F_2)^{k-i-1} (1 - F_1)^i & \text{if } k > i \end{cases} \quad (1)$$

where F_1 and F_2 indicate the expected probabilities of convergence at the k -th iteration during the learning phase and the practice phase, respectively; i is the number of iterations in the learning phase, and k is the count of write iterations. In the learning phase, the convergence probability is relatively small, which indicates more tries are necessary. After the learning phase, the write heuristic improves to have better convergence. This model was validated [6] and shown to match real-world experiments on write operations [15].

Our model: In this paper, we adopt Qureshi’s model and extend it to reflect that it takes a different average number of iterations to program a MLC to different digital values [16, 17]. Gray coding (GC) is also included in our design. For example, in a 2-bit MLC, the programming of a cell “00” (GC “00”) ends right after the RESET pulse. There is no need for SET pulses. Programming the cell to “11” (GC “10”) also ends faster as a slightly larger SET current can be applied. This slight overprogramming is safe as there is no subsequent resistance level that needs to be differentiated. The bottleneck for a 2-bit MLC write comes from programming “01” (GC “01”) and “10” (GC “11”), as shown in [16, 17]. In this paper, we assume the average number of iterations to program “01” (GC “01”) and “10” (GC “11”) are 8 and 6, respectively. This assumption is justified by previous device-level experimental work [16].

2.2 MLC PCM Read

Reading a PCM cell involves sensing its resistance level and mapping the analog resistance to a corresponding digital value. For SLC PCM, this capability is achieved by comparing cell resistance to a reference cell. The reference cell’s resistance is chosen as roughly the middle of the whole resistance range. In Figure 3(a), when compared with a $500K\Omega$ reference cell, a larger resistance indicates the stored value is “0”. Otherwise, the value is “1”.

A read operation for a MLC commonly uses binary search. Figure 3(b) shows that, to read a 2-bit MLC, the circuit first compares the resistance to a reference cell at $500K\Omega$. Depending on the comparison outcome, the circuit next compares the resistance to $50K\Omega$ or $5M\Omega$. It then determines the stored value. In general, a read operation for a n -bit MLC requires n iterations to complete.

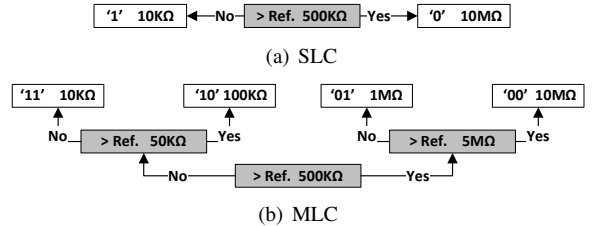


Figure 3. PCM read operation requires comparison to reference cells.

Given that reads are on the critical path, an aggressive design could perform all comparisons in parallel and complete the read in one iteration. For n -bit MLC PCM, parallel sensing needs to duplicate the read circuit ($2^n - 1$) times. This represents a trade-off between performance, increased sensing current and hardware overhead. As such, sequential sensing is widely adopted for MLC PCM read [7, 19].

3 Write Truncation and Form Switch

In this section, we first present some observations on the distribution of the number of iterations required in writing a MLC PCM line. Then, we develop two architectural innovations — *Write Truncation* (WT) and *Form Switch* (FS) — to improve the performance of MLC PCM writes and reads.

3.1 Observation: Uneven Distribution of MLC PCM Write Iterations

The staircase-up P&V MLC programming scheme uses multiple iterations to precisely program a PCM cell. Since each current pulse has the same width, the write latency is then determined by the number of iterations. When writing a PCM line, different cells require a different number of iterations. While the majority of cells likely finish in a modest number of iterations, some cells can take more iterations. Thus, the completion time of the whole line is determined by the cell that requires the largest number of iterations. In this paper, we define the cells that require more iterations as a *difficult-to-write* cell set.

Definition 3.1. *When writing a PCM line segment that has n cells, its **difficult-to-write** cell set is the subset of m cells that require the same or more iterations than the other cells. The cells in the difficult-to-write cell set require a certain number of iterations only for a particular write instance. Therefore, the difficult-to-write set membership may change from write to write.*

Definition 3.2. When writing k PCM line segments (each has n cells), the *difficult-to-write cell set* is all m *difficult-to-write cells* in each segment, i.e., it consists of $k \cdot m$ cells.

If n equals the number of cells in one PCM line, then the *difficult-to-write cell set* refers to the cells that globally require more iterations than other cells. If a PCM line is divided into k segments, then the *difficult-to-write cell set* refers to all m *difficult-to-write cells* in each segment. In this paper, we choose $n=64$ and $m=1$; k varies depending on the size of the PCM line.

The completion time of the whole line is determined by the cell that requires the largest number of iterations. Moreover, the *difficult-to-write cell set* and the most *difficult-to-write cell* vary from one write to another due to non-determinism in changing device resistance, e.g., the growth of filaments, as explained in Section 2.1. Hence, such a set cannot be predicted and precluded before or during an early stage of the write. For example, for the same PCM line, its 1st and 2nd cells are *difficult-to-write* in one write, while its 3rd and 4th cells may become *difficult-to-write* in the next write. Similarly, the same MLC line may require only two iterations in one write instance, but then it requires eight iterations in another write instance.

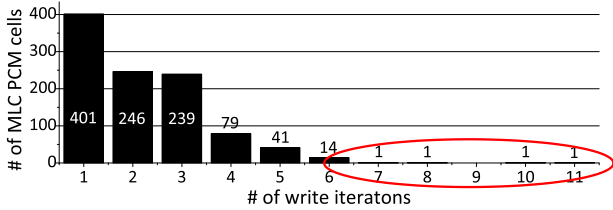


Figure 4. The distribution of the number of iterations for completing writing 1024 2-bit MLCs in a sample write.

Figure 4 plots the distribution of the number of iterations in writing a PCM line of 1024 2-bit MLCs, i.e., writing a 256-byte PCM line. This write finishes in 11 iterations. However, not all lines require this many iterations. Some may require fewer and others may require more. As discussed previously, a write to the same PCM line at different times may also require different number of iterations.

As Figure 4 shows, only one cell needs 11 iterations. The majority of cells finishes in less than 4 iterations. A few cells finish in 4-6 iterations. And only 4 cells require more than 6 iterations. When choosing ($n=1024$, $m=4$), these 4 cells form the *difficult-to-write cell set* for this write instance. If the excessive iterations can be avoided for the *difficult-to-write cells*, the line write latency can be reduced from 11 to 6 iterations, which is a 45% improvement for this line write instance¹.

¹Choosing ($n=1024$, $m=6$) would include these 4 cells and 2 other cells that require 6 iterations. For this particular example, removing 6 *difficult-*

3.2 Write Truncation

To prevent a few number of *difficult-to-write cells* from unnecessarily prolonging a write, we propose *write truncation* (WT) to truncate the trailing iterations of those cells. As a result, the *difficult-to-write cells* are *temporarily failed* since their writes did not finish. We apply single error correction, double error detection (SECDED) ECC to correct these soft errors. Note that the original line needs a mechanism to protect it from hard errors due to permanent cell damage. We assume *error correction pointer* (ECP) [20] is used, as illustrated in Figure 5(a). Hence, there are two sets of protection bits per PCM line: one for WT and the other for the baseline error protection.

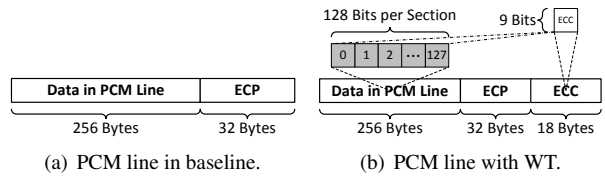


Figure 5. Adding ECC to PCM lines for write truncation (WT).

Figure 5(b) illustrates a line in our WT design. We choose ($n=64$, $m=1$), i.e., the *difficult-to-write cell set* consists of the cell that requires the largest number of iterations in each 64 cell (128-bit) segment. For a 256B PCM line, there are 32B of ECP (64b per 512b block as in the original design [20]), and 18B of SECDED ECC (9b per 128b block). Hence, the extra space overhead of WT is 18B per 256B line. We discuss how to mitigate the space overhead using *form switch* (FS) in Section 3.3.

The procedure of a write with WT is the following. When a line arrives, ECCs are computed, and written with the line using iterative P&V. At the end of each iteration, the write circuit checks in each 128-bit block (64 cells) how many cells are still being written. If there is only one cell left, the verification step tests if the current state is only one bit away from the target, such that ECC can cover this error. The write for this block is considered finished if the condition is true, and the full write can complete if all sub-writes are finished. Figure 6 summarizes our modifications to the existing P&V programming scheme. The SECDED ECC can rescue one bit per 128 bits (64 cells) and more write iterations are required if the condition is not satisfied. We adopt a gray code such that more write iterations bring the resistance closer to the target level without increasing the number of bit differences.

Discussion: One problem with WT is that if a hard fault occurred in a line, the faulty cell will become the dictating *difficult-to-write cell* since it can never be written no

to-write cells would have the same latency improvement as removing 4 cells.

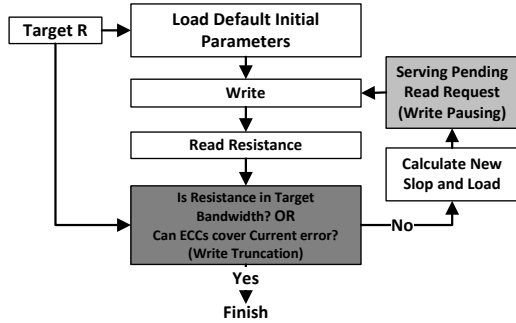


Figure 6. P&V programming with write truncation (WT).

matter how many iterations are used. This will deplete the opportunity of WT. Typically, when a cell cannot be programmed to the target resistance range in a preset maximum number of iterations (MAX), then the cell is identified as a hard fault which is rescued by ECP. However, WT may terminate the write before MAX is reached, so that the hard fault is covered by ECC, rather than ECP. This problem can be solved through wear-leveling techniques such as start-gap [21]. Wear-leveling will guarantee that periodically, all lines are swapped within a region. WT can be disabled during wear-leveling to permit the writing of cells with hard faults to reach MAX. This process guarantees that cells with hard faults are discovered and marked by ECP bits. ECC will only cover non-faulty cells, continuing to provide short write latencies in the presence of hard faults.

Unlike ECC in DRAM, which corrects soft errors, ECC for WT is not intended to cover soft errors—PCM is naturally resilient to them. DRAM ECC is designed to achieve a certain reliability level. It may fail with a slim probability that there are more errors than it can handle, while our ECC guarantees to correct one “soft error” (from a *difficult-to-write* cell).

The advantage of using ECC over other error-correction mechanisms, such as ECP, is that ECC can be generated regardless of the positions of *difficult-to-write* cells. This property is desirable because ECC can be generated *before* a write operation starts, which enables writing the data and ECC simultaneously. As a comparison, ECP uses pointers to record the locations of the cells to rescue. Since these locations can only be known when a write operation almost completes, ECP can only be generated at that time. Using ECP would actually prolong the write operation. Therefore, ECC is adopted in our design.

Figure 6 also illustrates a recent write optimization for MLC PCM — write pausing [6]. This optimization allows reads to preempt long-latency write operations to avoid being blocked for a long time. WT is orthogonal to *write pausing* as the latter does not reduce the bank level service time for each write. WT allocates more service time for reads

such that system performance is improved when memory accesses are intensive. With WT, MLC PCM write operations may have different programming iteration numbers and different write latencies. Without increasing the the complexity of memory controller, PCM chips can handle different write latencies with the new proposed framework and protocol in [22].

3.3 Form Switch

Although WT reduces the number of write iterations, it adds 9 bits per 128-bit PCM line segment, which corresponds to 7% storage overhead. In this section, we propose *form switch* (FS) to remove the storage overhead. FS compresses data to create space for storing ECC and transparently stores highly compressible lines in SLC form to reduce read latency.

FS employs *frequent pattern compression* [10] (FPC) to compress each MLC line. FPC examines every word and transforms frequently used patterns into fewer data bits and a prefix. A hardware compression unit is integrated in the memory controller to compress a line before it is stored in PCM. This compression unit also decompresses a line on a read. The experimental section shows that the compression unit’s latency, area, and energy overhead are negligible.

We use the following configurations in FS with FPC. The baseline has 256 bytes of MLC, with 32 bytes of ECP and 18 bytes of ECC. When a line is compressed with FPC, the write circuit generates 9-bit ECC for every 128-bit block of a *compressed* line. For example, suppose a line is compressed to 64 bytes, only 36 bits $(=(64 \times 8 / 128) \times 9)$ of ECC are generated, instead of the full 18 bytes. This saves space. The ECP bits are not affected.

After compression, if the total number of bits of the compressed data, prefix tag, and ECC is 128 bytes or fewer, then FS treats MLC as SLC. That is, every cell stores only one bit of information. This form switch not only speeds up writes, but also improves reads as only one comparison round to the reference resistance is performed. In addition, FS also benefits endurance as SLC experiences much less stress during writes than MLC. We use 1 bit per line as a flag to indicate the line form. When the bit is set, the PCM line is used as an SLC line. Otherwise, it is an MLC line.

If the total number of bits including data, tag and ECC is more than 128 bytes, then they are stored in MLC form, i.e., form switch is not used. In this case, there is a possibility for the total bit count to exceed 256B. If this happens, we borrow ECP bits since they are useful only when hard faults occur, as shown in Figure 7. Finally, if the ECP field is full and we cannot borrow bits to store ECC, then FS disables ECC protection, WT returns to a baseline write mechanism.

Discussion: To mitigate write imbalance within one line, intra-line rotation [20] evenly distributes writes to both data and ECP cells. We adopt the same design to balance writes

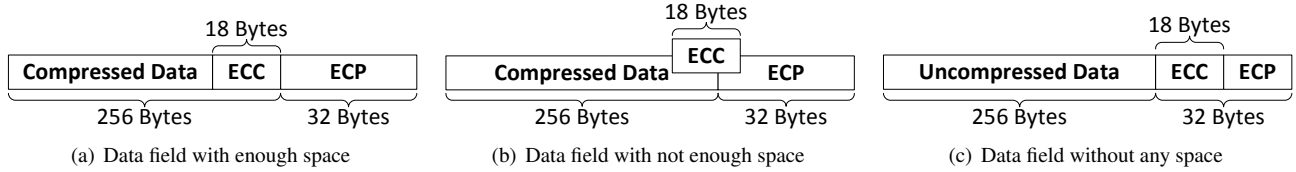


Figure 7. Integrated form switch with write truncation.

in one line. However, even with intra-line rotation, WT still tends to degrade PCM chip lifetime as including extra ECC bits increases the number of bits to be written. Compared to data bits, ECC bits are more likely to change as they are computed from data bits. We next study WT’s impact in more details. First, when adopting *differential write* [18], on average for each write, only about 15% data bits are changed and need to be written. Figure 8 compares the bit changes before and after adopting WT. MLC-C-M represents a scheme that compresses the data (no ECC) and stores it in MLC form. MLC-C-ECC represents FS + WT that compresses data: if the size of compressed data and ECC reduces to half, then lines are stored in SLC form; otherwise, the compressed data and ECC are stored in MLC form. As shown, there is a slight decrease for the bit changes — it is about 2% on average.

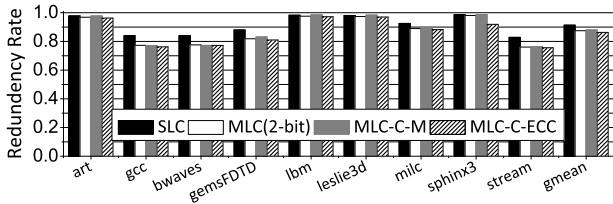


Figure 8. Write redundancy comparison when storing data in different forms.

In experiments, we observed on average FPC achieves 50% compression rate. However, approximately only 20% of all lines are highly compressible, i.e., the lines can be written in SLC form.

We also studied the lifetime impact after integrating the extra ECC. It was reported that storing data bits in compressed form reduces the total number of writes to the device, and thus, improves lifetime [13, 8]. Figure 9 compares the lifetime change after applying FS. Compared to the baseline without compression, FS also improves lifetime as fewer bits are changed. Compared to the compression only design (using FPC), FS tends to reduce lifetime as more bits are changed; however, highly compressible lines are stored in SLC form and thus tend to extend lifetime. The latter is based on the assumption that SLC has longer lifetime than MLC [7]. Similar studies for Flash showed that SLC has 10× better endurance than MLC [30]. Therefore, lifetime is helped from the 20% highly compressible lines and more benefits are expected when more lines are highly

compressible. In Figure 9, FSL_n indicates SLC cell’s lifetime is $n \times$ MLC cell’s lifetime. For FSL_{10} , we observed 7% lifetime degradation.

This degradation is modest for two reasons: (i) limited PCM chip lifetime is due to overprogramming [13]. While studies have reported up to 10^{12} cell endurance at the device level [23], only 10^8 or lower endurance can be achieved at the chip level. Various circuit level designs have shown great potential in mitigating overprogramming and extending chip lifetime, e.g., $27 \times$ longer lifetime using current provision [13]. (ii) Improving MLC write performance is much more difficult due to the precise control required for P&V programming. No current scheme can directly shorten MLC write latency. As shown in Section 5.2, our designs reduce MLC write latency by up to 57%.

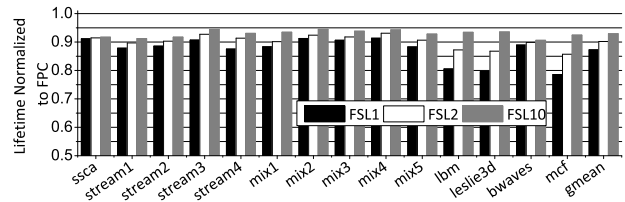


Figure 9. Lifetime degradation due to storing extra ECC bits (normalized to the scheme that adopts FPC compression).

4 Experimental Settings

We used Virtutech Simics (with the g-cache and trans-staller module) to conduct experiments. We assume an 8-core CMP system, which is similar to the architecture used in [6]. Like most proposals, the baseline has a small DRAM cache to filter accesses to PCM. Table 1 gives the baseline’s parameters. We modeled the cores in the CMP as single-issue, in-order pipelines to accelerate simulations. The baseline system’s DRAM cache is 256MB and located off chip. The DRAM cache uses a write-back policy with LRU replacement. It mitigates the endurance problem by filtering write operations. The DRAM cache also merges hot writes, which significantly reduces the write traffic to the PCM.

The baseline has a 32GB PCM memory that consists of 4 individual memory ranks. Each rank has 8 banks with a 32-entry write queue (WRQ) and an 8-entry read queue (RDQ). The PCM uses *differential writes* to store a value only when the new value differs from the old one in a PCM

System	8-core CMP, 4GHz
Processor Core	single issue in-order, 32KB iL1, 32KB dL1
L2 (private)	2MB, 4-way, LRU, 64B/256B line size, write back
DRAM Cache (private)	32MB, 8-way, 64B/256B cache line size, 50ns (200 cycles) access latency, write back
Main Memory	32GB PCM, 4KB page, 4 ranks of 8-bank each, 32 entries write queue/bank, 8 entries read queue/bank, read priority scheduling (unless WRQ is > 80% full), with differential write support
PCM Latency	SLC/MLC read: 125/250ns (500/1000 cycles); write iteration: 250ns (1000 cycles); WT: 9b ECC covers 128b/section; write model: $F_1 = 0.375$, $F_2 = 0.625$, $i = 2$ for '01', averagely 8 iterations; $F_1 = 0.425$, $F_2 = 0.675$, $i = 2$ for '10' (GC '11'); averagely 6 iterations; fixed 1 iteration for '00', fixed 2 iterations for '11' (GC '10')

Table 1. Baseline Configuration

cell. When a bank is idle, the oldest request in the highest priority queue is issued to the bank. In general, the read queue has a higher priority than the write queue. However, when the write queue is 80% full, its writes are serviced ahead of reads. We assume the read latency of SLC PCM and MLC PCM are 500 cycles and 1000 cycles, respectively. We determined that one write iteration is 1000 cycles from [6]. A write operation takes a dynamic number of iterations to finish. The average write latency is ~ 8000 cycles, which is the same as [6].

We chose a set of memory intensive benchmark programs. These programs include *lbm*, *leslie3d*, *bwaves*, *mcf* from SPEC2006, *ssca* from HPC Graph Analysis [24] and a stream workload STREAM [25]. Table 2 summarizes the program’s memory access behaviors (read MPKI and write MPKI). We executed the benchmarks in multi-programmed mode; each core executes the same benchmark in a private space. In addition, we evaluated the performance of mixed benchmark workloads. One mixed workload contains 4 different programs (2 copies of each).

To compare the effectiveness of different schemes, we use the following criteria. We define *effective read latency* as following:

$$\text{Effective_Read_Latency} = t_{rdready} - t_{rin}, \quad (2)$$

where $t_{rdready}$ is the time when the PCM bank finishes servicing a read operation; and t_{rin} is the time when the read operation reaches the read queue.

We define *effective write latency* similarly,

$$\text{Effective_Write_Latency} = t_{wdready} - t_{win}, \quad (3)$$

where $t_{wdready}$ is the time when the write operation releases its corresponding PCM bank; and t_{win} is the time when the write request enters the write queue.

The *speedup* metric is defined as

$$\text{Speedup} = \frac{CPI_{baseline}}{CPI_{tech}}, \quad (4)$$

Benchmark	Description	RPKI	WPKI
ssca_m	8 copies of ssca	7.3	3.58
mcf_m	8 copies of mcf	6.59	2.88
bwaves_m	8 copies of bwaves	10.65	4.39
lbm_m	8 copies of lbm	8.91	4.05
leslie3d_m	8 copies of leslie3d	6.7	2.32
stream1_m	8 copies of COPY	9.94	3.28
stream2_m	8 copies of SCALE	9.91	4.9
stream3_m	8 copies of SUM	8.6	4.2
stream4_m	8 copies of TRIAD	11.15	3.64
mix_1	2lbm-2leslie3d-2mcf-2COPY	7.9	3.1
mix_2	2lbm-2leslie3d-2COPY-2TRIAD	9	2.25
mix_3	2lbm-2mcf-2COPY-2TRIAD	8.98	3.43
mix_4	2leslie3d-2mcf-2COPY-2TRIAD	8.36	2.99
mix_5	2lbm-2leslie3d-2mcf-2TRIAD	8.13	3.15

Table 2. Simulated Applications

where $CPI_{baseline}$ indicates the CPI of the baseline machine, and CPI_{tech} is the CPI with a proposed technique. The same speedup metric is used in [6]. It is also similar to the *weighted speedup* in [26]. We compared the following schemes in our experiments.

- **WP** — write pausing in [6]. It is enhanced with *differential write*.
- **WP+FS** — write pausing with form switch only (no extra ECC bits). This compares our design to compression-only designs [8, 13].
- **WP+WT** — write pausing and write truncation.
- **WP+WT+FS** — write pausing, and write truncation and form switch (with extra ECC bits).

To evaluate the impact on PCM chip lifetime, we collected memory write traces and did our analysis on these traces assuming effective wear leveling [21].

5 Evaluation

5.1 WT and FS Implementation Overhead

We synthesized WT and FS in VHDL and report the latency, energy and area overhead in Table 5.1. The overhead of WT comes from introducing extra ECC bits and the overhead of FS comes from performing FPC compression. Given that MLC PCM read latency is 250ns, the latency overhead of WT and FS is less than 1% per access. The energy overhead is less than 3 pJ per 64B line segment, which is negligible as reading and writing PCM consumes 2 pJ/bit and 13 pJ/bit respectively [18] and one line segment contains 512 bits. The area overhead is comparable to the area of a 16KB PCM when the PCM line is 256 bytes. This 16kb is negligible compared to the 32GB PCM in the baseline.

Operation	WT	FS	
	En/Decoding	Compress	Decompress
Latency	0.5 ns	0.7 ns	1.2 ns
Energy (64B section)	0.7 pJ	1.2 pJ	2.1 pJ
Area (64B section)	14300 μm^2	7300 μm^2	18500 μm^2

Table 3. The Latency, Energy and Area Overhead of WT and FS

5.2 Effective Write Latency

Figure 10 compares the effective write latency using different schemes for 256B line size. We normalized the results to the baseline. We report the results for 64B and 256B PCM line sizes. Current systems often adopt 64B line sizes. However, with increasing capacity in the last level cache (LLC), future systems are likely to adopt large line sizes to reduce tag overhead. For example, IBM’s recently announced zEnterprise [27] uses a 256B LLC line size.

For a 256B line, on average, write pausing (WP) experiences a 3% latency increase since some writes are preempted and take more time to complete. Adopting FS, i.e., WP+FS, decreases latency by 17% over the baseline as writing a line in SLC form is much faster than in MLC form. The latency of compression and ECC encoding contributes less than 1% overhead.

From Figure 10, write latency is reduced due to write truncation (WT). WT removes the iterations that work on *difficult-to-write* cells as long as these cells can be corrected using ECC bits. WT and WT+FS reduce write latency to 59% and 44% of the baseline, respectively. An interesting observation is that WP+WT+FS’s reduction over WP+WT is about the same as WP+FS’s reduction over WP, which indicates WT and FS are orthogonal in reducing effective MLC write latency.

With 64B line size, we observed a larger latency increase (i.e., 10%) for WP as more writes are preempted. WP+FS has a smaller latency reduction than the 256B line setting. The reason is that the opportunity to compress a shorter line by 50% or more is smaller. WT becomes more effective as more write instances benefit from reduced write iterations.

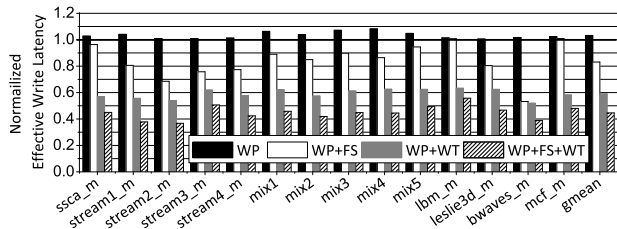


Figure 10. Effective write latency (256B).

5.3 Effective Read Latency

Figure 11 compares the effective read latency using different schemes for 256B line size. On average, WP improves read latency to 69% of the baseline. Since WT im-

proves bank service time, it is orthogonal to WP — adding WT further reduces the read latency to 59% of the baseline. FS improves read latency by converting accesses to highly compressible MLC lines to SLC accesses. This conversion is done at the architecture level, which is transparent to upper levels, including the OS and user applications. When WP, WT and FS are applied together, read latency is dramatically reduced to 51% of the baseline, or 74% of WP’s read latency. In some cases, e.g., `bwaves_m` and `stream2_m`, we found FS by itself is more effective than WT. This situation arises because more lines are highly compressible and converting them to SLC form reduces read access latency.

While both WT+FS and FS benefit from accessing faster SLC cells, WT+FS is 18% better than FS. This result happens because write pausing is disabled if a write operation finishes more than 80% of one iteration. By reducing the write latency with WT, read operations have fewer chances to be blocked by nearly-finished writes, and thus, they wait less time in the queue. This further improves read latency in WT+FS.

When the PCM line size increases from 64B to 256B, fewer write operations are issued. We observed larger read latency reduction as WP becomes more effective for 256B and helps all schemes.

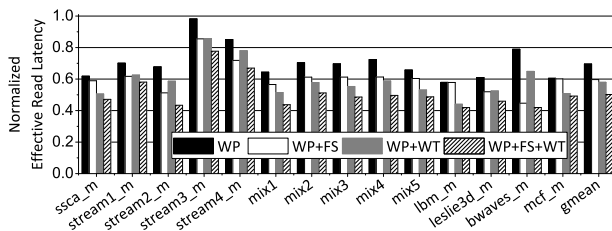
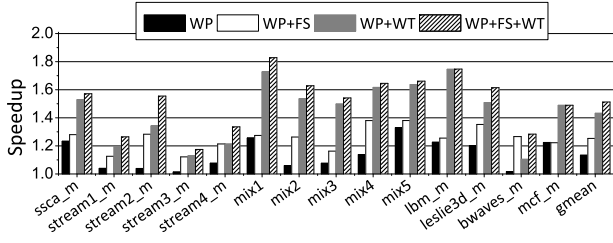


Figure 11. Effective read latency (256B).

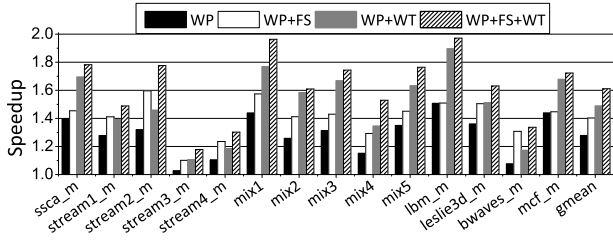
5.4 Performance Analysis

By reducing both write and read latency, our designs improve program performance. The results are summarized in Figure 12. For a 256B PCM line, WP has a 27% performance improvement due to faster critical read operations. This result is slightly worse (but comparable) to the result reported in [6]. Our performance results for WP differ from the original work slightly because we used *differential writes* to remove redundant cell writes, which increases the chance (by a small amount) that reads are blocked by writes. WP does not achieve a noticeable performance improvement for `stream3_m`. In this program, the write request ratio is high and the write requests come in a burst. WP is less beneficial in this situation.

Figure 12 compares WP, WP+WT, and WP+WT+FS. The 256B results show that WP+FS and WP+WT improve performance by 10% and 16% over WP respectively, and WP+WT+FS gains another 9% performance over WP+WT.



(a) 64B PCM line



(b) 256B PCM line

Figure 12. The IPC comparison of different schemes (normalized to the baseline).

In total, WP+WT+FS has a 26% average performance improvement over WP, indicating WT and FS are orthogonal.

When the PCM line size increases from 64B to 256B, we observed 5% better performance due to better performance from WP, as discussed in Section 5.2 and 5.3.

5.5 WT Section Size

In our design, we use 9 bits per 128 consecutive bits SECDED ECC. A 256-byte PCM line is divided into 16 128-bit segments and requires 18-byte ECC that can rescue up to 16-bit *difficult-to-write* cells. However, if two *difficult-to-write* cells appear in the same segment, then the write operation cannot be terminated early as it is beyond the correction capability of our ECC. The write circuit knows the number of *difficult-to-write* cells at the end of each write iteration such that no error may exist as not correctable.

It is also possible to adopt a more powerful ECC, e.g., 8EC9ED (8-bit error correction and 9-bit error detection) BCH ECC for all bits in a 256-byte PCM line. This code is a *global rescue* design. Table 4 compares the number of ECC bits required for each PCM line, the latency, and the area of three ECC codes. A 128-bit segmented ECC requires 0.5ns [29]. However, the *global rescue* scheme introduces non-negligible latency and area overhead.

Scheme	Type	Size	Latency	Area
64 bits section	SECDED	32B	0.5ns	$\sim 0.01 \text{ mm}^2$
128 bits section	SECDED	18B	0.5ns	0.0143 mm^2
global rescue	8EC9ED	12B	30 - 80 ² ns	1 mm^2

Table 4. The Comparison of Different WT Section Sizes

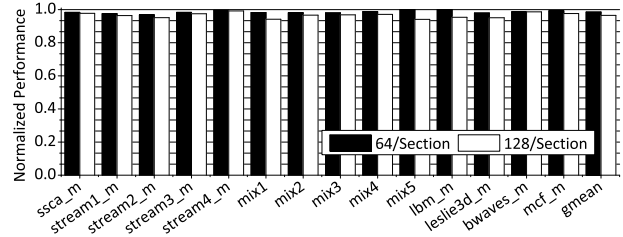


Figure 13. The comparison of IPC using different ECC codes.

We evaluated the performance impact of using SECDED. Figure 13 compares the performance of three codes. We normalize the results to *global rescue*. In our simulation, we did not count the encoding and decoding delay of *global rescue*. The performance differences between *global rescue* and the other two schemes depends only on the ability to cover *difficult-to-write* cells. 64-bit SECDED has a 2.1% performance improvement over 128-bit SECDED. However, it requires 32 bytes per line while 128-bit SECDED requires only 18 bytes.

5.6 Iteration Latency

In previous experiments, we set the average write latency to be the same as [6], which used 1000 cycles per iteration. The iteration latency depends on material characteristics and technology advances. We varied the latency from 500 CPU cycles to 1500 CPU cycles. Figure 14 summarizes the change in IPC as this latency is changed. From the figure, WT + FS + WP has higher speedup with longer iteration latencies. Our schemes, WT + FS, win 33% performance improvement over WP. With a small iteration latency, a write requires a shorter bank service time, which leaves more service time for reads. Therefore, removing write latency and the bank busy time has less benefit for read operations as iteration latency is decreased, and a diminishing improvement of program performance. But WT + FS + WP still gain extra 13% performance over WP.

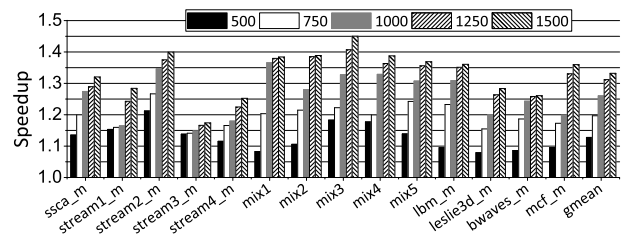


Figure 14. IPC comparison of WP+WT+FS normalized to WP with varying write iteration latencies (256B).

²To exam the covering power of global rescue, we used 0 in simulations.

6 Prior Art

While PCM has many advantages over traditional DRAM, it suffers from limited endurance and long latency write. The endurance problem is mitigated by removing unnecessary bit-level writes [18], and adopting wear leveling [21] and salvaging [20]. Recent work on salvaging [28] integrates 5EC-6ED BCH code into memory line and prevents PCM cell from both hard faults and soft errors introduced by peripheral circuit. BCH code is fast in rescuing first 2 bit errors and thus good for rarely appeared soft errors. For proactively introduced soft errors in our setting (up to 16 bit errors due to WT), correcting error using BCH code is very slow and introduces significant overhead.

There are different approaches to address the long latency write problem. By merging frequent writes within the DRAM cache, the number of writes issued to a PCM device can be greatly reduced [31]. Write pausing [6] allows performance critical read operations to preempt writes. Comparing to morphable memory system [7] that also converts MLC lines to SLC lines, FS is data-centric and transparent to higher system layers, including the operating system.

Compression has been applied in PCM to reduce the total number of bits writing to the cells. Compressed data with a smaller size can be written into fewer PCM cells than original data. Comparing to compression-only designs [8, 13], FS stores highly compressible lines in SLC form using the same location such that it reduces the read accesses while [8] target at improving lifetime and energy only.

7 Conclusions

We propose *Write Truncation* (WT) and *Form Switch* (FS) to speedup write and read operations in MLC PCM. WT uses SECDED ECC to avoid fully writing a small set of *difficult-to-write* cells. It terminates write operations for these cells early to improve write performance. FS compresses a PCM line to mask the space overhead required by WT. It also helps improve read performance by storing data in SLC form. Our techniques improve write latency by 57% and read latency by 28%. These gains translate into a 26% average performance improvement over existing state-of-the-art techniques on a range of multi-programmed workloads in an 8-core chip multiprocessor system.

8 Acknowledgments

We thank the anonymous referees for their valuable comments and suggestions. We also acknowledge the support from PCM@Pitt research group. This research is supported by National Science Foundation grants CNS CAREER-0747242, CNS-1012070, CCF-0811295 and CCF-0811352.

References

[1] S. Borkar, "Thousands Core Chips - a Technology Perspective," in *DAC*, 2007.

- [2] C. Lefurgy et al., "Energy Management for Commercial Servers," in *IEEE Computer*, 2003.
- [3] The ITRS Report, 2009. <http://www.itrs.net/>
- [4] F. Bedeschi et al., "A Bipolar-selected Phase Change Memory Featuring Multi-level Cell Storage," in *JSSC*, 2009.
- [5] S. Raoux et al., "Phase-Change Random Access Memory: a scalable technology," in *IBM J. Res. Dev.*, 2008.
- [6] M.K. Qureshi et al., "Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing," in *HPCA*, 2010.
- [7] M.K. Qureshi et al., "Morphable Memory System: a Robust Architecture for Exploiting Multi-level Phase Change Memories," in *ISCA*, 2010.
- [8] G. Sun et al., "A frequent-value based PRAM memory architecture," in *ASP-DAC*, 2011.
- [9] A. Cabrini et al., "Voltage-driven Multilevel Programming in Phase Change Memories," in *IEEE MTTT*, 2009.
- [10] A.R. Alameldeen and D.A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *ISCA*, 2004.
- [11] D. Mantegazza et al., "Statistical Analysis and Modeling of Programming and Retention in PCM Arrays," in *IEDM*, 2007.
- [12] M. Boniardi et al., "Impact of Material Composition on the Write Performance of Phase-Change Memory Device," in *Intl. Memory Workshop*, 2010.
- [13] W. Zhang and T. Li, "Characterizing and Mitigating the Impact of Process Variations on Phase Change based Memory Systems," in *MICRO*, 2009.
- [14] L. Jiang et al., "Enhancing Phase Change Memory Lifetime through Fine-Grained Current Regulation and Voltage Upscaling," in *ISLPED*, 2011.
- [15] T. Nirschl et al., "Write Strategies for 2 and 4-bit Multi-level Phase-Change Memory," in *IEDM*, 2007.
- [16] A. Pantazi et al., "Multilevel PCM Modeling and Experimental Characterization," in *E-PCOS*, 2009.
- [17] M. Joshi et al., "Mercury: A Fast and Energy-Efficient Multi-level Cell based Phase Change Memory System," in *HPCA*, 2011.
- [18] P. Zhou et al., "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.
- [19] A.C. Calligaro et al., "Comparative Analysis of Sensing Schemes for Multilevel Non-volatile Memories," in *ICISS*, 1997.
- [20] S. Schechter et al., "Use ECP, not ECC, for Hard Failures in Resistive Memories," in *ISCA*, 2010.
- [21] M. K. Qureshi et al., "Enhancing Lifetime and Security of PCM-based Main Memory with Start-Gap Wear Leveling," in *MICRO*, 2009.
- [22] K. Fang et al., "Memory Architecture for Integrating Emerging Memory Technologies," in *PACT*, 2011.
- [23] K. Kim and S. Ahn, "Reliability Investigations for Manufacturable High Density PRAM," in *IRPS*, 2005.
- [24] HPCS Scalable Synthetic Compact Applications #2 Graph Analysis. <http://www.graphanalysis.org/benchmark/>
- [25] STREAM. <http://www.cs.virginia.edu/stream/>
- [26] D.M. Tullsen and J.A. Brown, "Handling Long-latency Loads in a Simultaneous Multithreading Processor," in *MICRO*, 2001.
- [27] J. Warnock et al., "A 5.2Ghz Microprocessor Chip for the IBM zEnterprise System," in *ISSCC*, 2011.
- [28] Doe Hyun Yoon et al., "FREE-p: Protecting non-volatile memory against both hard and soft errors", in *HPCA*, 2011.
- [29] R. Naseer, and J. Draper, "DEC ECC design to improve memory reliability in Sub-100nm technologies," in *ICECS*, 2008.
- [30] J. Thatcher et al., "Nand flash solid state storage for the enterprise: An in-depth look at reliability," in *SNIA*, 2009.
- [31] M.K. Qureshi et al., "Scalable High Performance Main Memory System using Phase-Change Memory Technology," in *ISCA*, 2009.