SD-PCM: Constructing Reliable Super Dense Phase Change Memory under Write Disturbance

Rujia Wang
University of Pittsburgh
ruw16@pitt.edu

Lei Jiang
University of Pittsburgh
lej16@pitt.edu

Youtao Zhang
University of Pittsburgh
zhangyt@cs.pitt.edu

Jun Yang
University of Pittsburgh
juy9@pitt.edu

Abstract

Phase Change Memory (PCM) has better scalability and smaller cell size comparing to DRAM. However, further scaling PCM cell in deep sub-micron regime results in significant thermal based write disturbance (WD). Naively allocating large inter-cell space increases cell size from $4F^2$ ideal to $12F^2$. While a recent work mitigates WD along word-lines through disturbance resilient data encoding, it is ineffective for WD along bit-lines, which is more severe due to widely adopted μTrench structure in constructing PCM cell arrays. Without mitigating WD along bit-lines, a PCM cell still has $8F^2$, which is 100% larger than the ideal.

In this paper, we propose SD-PCM for achieving reliable write operations in super dense PCM. In particular, we focus on mitigating WD along bit-lines such that we can construct super dense PCM chips with $4F^2$ cell size, i.e., the minimal for diode-switch based PCM. Based on simple verification-n-correction (VnC), we propose LazyCorrection and Pre-Read to effectively reduce VnC overhead and minimize cascading verification during write. We further propose (n:m)-Alloc for achieving good tradeoff between VnC overhead minimization and memory capacity loss. Our experimental results show that, comparing to a WD-free low density PCM, SD-PCM achieves 80% capacity improvement in cell arrays while incurring around 0-10% performance degradation when using different (n:m) allocators.

ASPLOS '15, March 14–18, 2015, Istanbul, Turkey..

http://dx.doi.org/10.1145/2694344.2694352

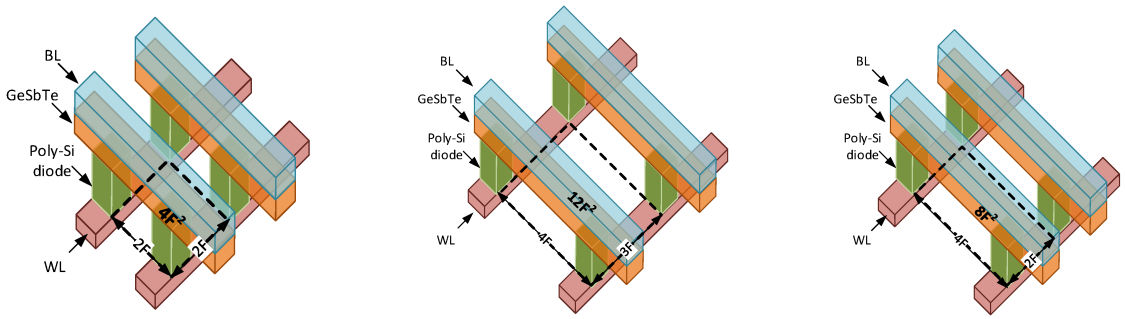
Categories and Subject Descriptors C.0 [*Computer Systems Organization*]: GENERAL-Hardware/software interfaces

Keywords Phase Change Memory; Write Disturbance

1. Introduction

Phase Change Memory (PCM) has recently emerged as a promising memory technology that supplements to traditional DRAM technology for future main memory systems. As modern applications increasingly demand for large amount of memory, it is important to construct highly scalable and dense main memory with low power consumption [14, 21]. Unfortunately, DRAM faces both scalability and power consumption challenges — the path to scale DRAM beyond 20nm is unclear, and the refresh power from DRAM chips of large capacity is significant [16]. PCM has better scalability and better cell size — PCM chips can be constructed with 20nm technology and diode-switch based PCM exhibits $4F^2$ cell size [8]. As a non-volatile memory technology, PCM has no leakage power from the cell. In recent studies, PCM has been proposed to replace a significant portion of DRAM main memory for achieving the overall energy effectiveness [14, 21, 35].

A PCM cell takes advantage of phase change material (GST [24]) to record data. By the application of heat generated through electrical pulses, the phase change material inside each PCM cell can be melted (RESET) or crystallized (SET). Scaling PCM in deep sub-micron regime faces non-negligible inter-cell thermal interference during programming, referred to as write disturbance (WD) phenomenon. That is, the heat generated for writing one cell may disseminate beyond this cell and disturb the resistance states of its neighboring cells. WD in PCM was first observed at 54nm [15], and has become more significant at and below 20nm technology [4, 13].



(a) Ideal super dense chip ($4F^2$/cell) (b) Prototype chip [8] ($12F^2$/cell) (c) DIN[10]-enhanced chip ($8F^2$/cell)

Figure 1. Write disturbance (WD) has a large impact on PCM cell density.

The existing solutions for mitigating WD in PCM are ineffective. A simple yet widely adopted solution is to allocate sufficient thermal bands between cells, which results in significantly deviated PCM chip capacity from its theoretical ideal. For example, in recent prototype PCM chips [4, 8], 20nm and 40nm extra inter-cell space was introduced along word-lines and bit-lines, respectively, which results in only 33% memory capacity of the ideal, as shown in Figure 1(b). Naively adopting mitigation techniques developed for other memory technologies (such as NAND Flash), e.g., verify-n-correct (VnC), may lead to cascading verification and large performance overhead.

The work that is most related to our design is *DIN* [10], a recent proposal that addresses WD in PCM through disturbance-aware data encoding, which minimizes data patterns that are more vulnerable to disturbance. However, *DIN* applies only to WD along word-lines. Studies revealed that WD errors along bit-lines are more significant than those along word-lines. This is because PCM cell array widely adopts μTrench structure [18], which connects cells along one bit-line with one GST rail while different bit-lines are isolated by dielectric, e.g., oxide material. Since oxide has better thermal isolation [12] than GST, the disturbance along bit-line is more significant than that along word-line. Without proper WD mitigation along bit-lines, *DIN* achieves $8F^2$ cell size, an 33% improvement over the low density design. However, it is still 100% larger than the $4F^2$ ideal, , as shown in Figure 1(c).

In this paper, we propose SD-PCM for achieving reliable write operations in super dense PCM. A super dense PCM chip is built by allocating only minimal inter-cell space along both bit-lines and word-lines. We devise a set of techniques to mitigate WD errors with low performance overhead. In particular, we make the following contributions.

- We illustrate that WD along bit-lines translates to disturbance errors with high possibility. A simple adaption of verification and correction (VnC) for WD errors would require VnC on per-write basis and may lead to cascading verification, resulting in large performance degradation. In the paper, we propose LazyCorrection and Pre-Read that effectively minimize verification and correction overhead, respectively.
- We propose (n:m)-Alloc for achieving good runtime tradeoff between VnC overhead minimization and memory capacity loss. (n:m)-Alloc selectively marks a set of device rows as *no-use*, i.e., not to be allocated to any process. By storing no data in these rows, we effectively reduce the necessity of doing verification on these rows, and thus gain the flexibility to match the VnC overhead to the performance demand of high priority applications.
- We evaluate our proposed schemes and compare them to the state-of-the-art. The experimental results show that our design can achieve 80% capacity improvement in memory cell arrays over the WD-free low density PCM design. SD-PCM incurs around 0-10% performance degradation when adopting different (n:m) allocators.

The rest of this paper is organized as follows. Section 2 introduces PCM and WD background. Section 3 illustrates the challenges of mitigating WD along bit-lines. Section 4 elaborates the design details of our three schemes. Section 5 and Section 6 discuss our experimental methodology and results. Section 7 discusses additional related works. Section 8 concludes the paper.

2. Background

2.1 PCM Basics

Phase Change Memory (PCM) stores data by utilizing phase change material, e.g., $Ge_2Sb_2Te_5$ (GST). A PCM cell uses its fully amorphous state with high resistance and its fully crystalline state with low resistance to represent bit '0' and '1', respectively. PCM switches between two states by heating the cell with current pulses. The RESET operation heats the cell above melting temperature (about 600 °C) and quench quickly; the SET operation requires the temperature above the crystallization temperature (about 300 °C) and below the melting temperature for a relatively long period. In

short, SET requires lower temperature but longer time than RESET.

A PCM chip can be built in 2D cell array that is similar to DRAM [6, 24]. As shown in Figure 1(a), a diode-switch based PCM cell can achieve $4F^2$ per cell (where F is the feature size) [4, 8]. The cell that stores one bit is called Single-level-cell (SLC) while the cell that stores multiple bits is called Multiple-level-cell (MLC).

2.2 Write Disturbance in PCM

The write disturbance (WD) in PCM arises from the inter-cell heat dissemination during programming. When resetting a PCM cell, the programming heat may spread to its neighboring cells and corrupt their stored values. Figure 2 exhibits two neighboring PCM cells along one bit-line. Since these two cells belong to different memory lines (word-lines), a write happens on at most one of them. For example, one write only reset the left cell and leave the right cell untouched. The right cell is referred to as an *idle* cell in this paper. Because of the disseminated heat along the bit-line, in the case if the right cell is in fully amorphous state, a part of its amorphous volume may collapse into crystalline fractions, which greatly reduces its resistance and loses the stored bit '0' [4, 13].

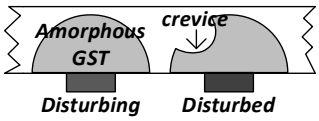


Figure 2. WD along a bit-line.

WD in PCM was first reported at 54nm [15]. However, as fast technology scaling keeps reducing inter-cell distance, the WD in PCM has become a non-negligible reliability issue at 20nm. Recent works [4, 13] demonstrated that WD is a major obstacle for the practical deployment of PCM.

2.2.1 WD Vulnerable Data Pattern

WD in PCM may manifest as an error only in a particular data pattern, i.e., the bit in a PCM cell $Cell_1$ may be disturbed only when its neighboring cell $Cell_0$ is under RESET while $Cell_1$ is idle and in amorphous state. This is because: (i) heat dissemination decays fast horizontally. The temperature that $Cell_1$ can reach (due to $Cell_0$ being RESET) is always below the melting temperature threshold but may be higher than the crystalline temperature threshold. Thus, $Cell_1$ may be crystallized if it stays in RESET state (i.e., amorphous state), but cannot be melt if it is in SET state (i.e., crystalline state) [13]. (ii) since SET current is only about half of RESET current, the temperature increase during SET is four times lower than that during RESET [26]. Thus, the disturbing influence of SET operation can be ignored [27].

Figure 3 illustrates the location of vulnerable PCM cells. For a cell being RESET, its neighbor cell is vulnerable if it stores bit 0 and is idle. Vulnerable cells may appear along a word-line or a bit-line. Vulnerable cells do not always produce disturbance errors. However, in the worst case, one RESET may disturb four neighboring cells and introduces four bit errors.

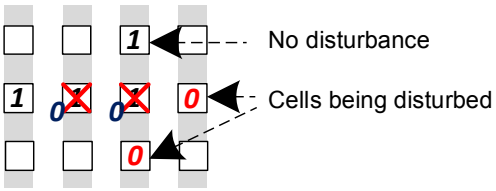


Figure 3. WD-vulnerable data pattern.

2.2.2 Modeling Write Disturbance

We adopted the WD model from [10] that includes a PCM cell thermal model, a PCM cell scaling model, and a PCM thermal disturbance model. The PCM cell thermal model is used to evaluate the interference temperature elevation during one RESET operation for the cells generated using the PCM cell scaling model. By combining the PCM cell thermal model and the PCM cell scaling model, the disturbance temperature increment can be computed for different technology nodes. In this paper, we focus on 20nm technology. We fed the disturbance temperature rise during a RESET operation into the PCM thermal disturbance model to compute WD bit error rate.

Table 1. Disturbance Probability for $4F^2$ cells

Between two cells along	Temp↑	Error rate (SLC)
Word-line	310 °C	9.9%
Bit-line	320 °C	11.5%

Given feature size F (F=20nm in this paper), 2F cell-to-cell space represents the minimal distance, i.e., two cells are placed next to each other. The cell array often adopts the μTrench structure [18], where PCM cells are fabricated on a GST (chalcogenide) bar serving as a bit-line. Two GST strips are separated by oxide material. Since oxide has better thermal isolation [12] than GST, the interference temperature elevation along a word-line for 40nm inter-cell space is 310 °C [10], and along a word-line for 40nm inter-cell space is 320 °C, i.e., 10°C higher [4]. We summarize the thermal disturbance error rates for SLC PCM cells in Table 1. The details of the models are in [10].

2.2.3 Difference from Other PCM Errors

Write disturbance error is different from other types of errors in PCM, such as hard failure error and resistance drift error. The error rate of write disturbance is much higher than the rates of these two types of errors.

The hard failure error in PCM happens when a PCM cell fails to change its state when injecting programming currents. A PCM cell may stuck at high or low resistance state and its value can not be changed any more. In contrast, WD error is a type of soft errors — a disturbed cell can be corrected by rewriting.

The soft error due to resistance drift in PCM [5, 31] reflects the fact that the resistance of a PCM cell increases spontaneously after write operation. It happens when there is no operation accessing the cell. In addition, resistance drift happens even at room temperature [9]. Studies show that MLC PCM cells are vulnerable to resistance drift. Resistance drift is not treated as a major reliability concern in this paper as we consider SLC PCM only. In contrast, WD error happens when an amorphous state cell is idle but its neighboring cell is under RESET. Moreover, it can only occur under a higher disturbing temperature (i.e., above crystallization point). The disturbed cell can be considered as "partially" SET.

3. Motivation

In this section, we discuss the inadequacy of two widely adopted disturbance mitigation techniques. One is to allocate sufficient inter-cell space. The other is to rectify disturbed cells through Verify-n-Correct (VnC).

3.1 Allocating Large Inter-cell Space

WD in PCM heavily depends on the inter-cell distance. The shorter distance between two PCM cells there is, the more significantly WD errors prevail [13, 25]. Therefore, a simple yet widely adopted approach to mitigate write disturbance is to allocate sufficiently large inter-cell space.

Figure 1(b) illustrates a WD-free prototype chip with $12F^2$ cell size [8]. Thermal disturbance decays very fast horizontally. As a result, allocating 3F along bit-line and 4F along word-line effectively eliminates all WD errors. The chip allocates more inter-cell space along bit-lines because WD along bit-lines is more severe than that along word-lines. The major issue of large inter-cell space is memory capacity loss, e.g., the cell array of the prototype chip achieves only 33% of the memory capacity of an ideal cell array ($4F^2$ cell size).

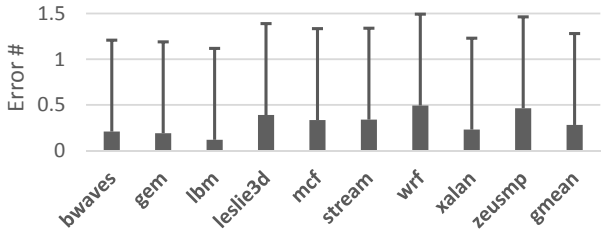
Figure 1(c) illustrates the cell array enhanced with WD-aware data encoding scheme [10]. It shrinks inter-cell space along word-lines but not along bit-lines. As a result, WD may manifest along word-lines only. The DIN scheme [10] performs additional checks and rewrites to ensure write reliability. The DIN-enhanced cell array achieves $8F^2$/cell, or a 33% memory capacity increase from that in the prototype chip. However, comparing to the ideal, its cell size is still 100% bigger, representing 50% capacity loss.

Note, the cell array density improvement often translates to a smaller chip size reduction. For example, PCM cells occupy 46.6% of the total chip area of the prototype chip [8]. The DIN-enhanced scheme achieves 33% cell array density improvement, or 15.4% chip size reduction.

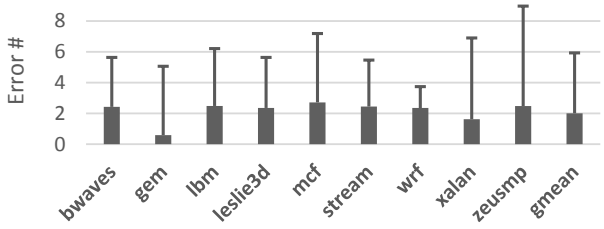
Challenge 1: is it possible to enable the super dense PCM with or close to $4F^2$ cell size?

3.2 VnC: Verify and Correct Disturbance Errors

While WD errors are soft errors, traditional software correction schemes, such as SECDED ECC or strong BCH, are not applicable. This is mainly due to the relatively high WD error rates (as shown in Table 1).



(a) Manifested errors within the same word-line



(b) Manifested errors in one adjacent PCM line

Figure 4. WD errors when writing a PCM line in super dense PCM ($4F^2$/cell).

Figure 4 reports the average and maximum WD errors when writing a PCM line.We adopt differential write [35] to minimize the total number of cell writes per write operation, and the DIN scheme [10] to mitigate WD along word-lines. From the figure, the WD errors within the same word-line are well mitigated — on average, there are 0.4 errors. Unfortunately, one write produces up to 9 WD errors in one adjacent line (64B). This requires 82 bits, or 16% space overhead if adopting a strong BCH code [32]. In addition, WD errors accumulate fast. Writing a PCM line ten times may produce 20 WD errors in different bit locations of its adjacent line, defeating a strong ECC protection with reasonable space overhead.

Another widely adopted disturbance mitigation approach is to verify and, if necessary, correct appeared WD errors, this is referred to as VnC in the paper. The issue with WD errors along bit-lines is that these errors are not in the PCM line being written. In order to apply VnC scheme, we need to read the old data of adjacent device lines before writing the current line, read again after write, compare the data from two reads, and correct adjacent lines if WD errors appear. Disturbed cells are in '0' state and require RESET operations

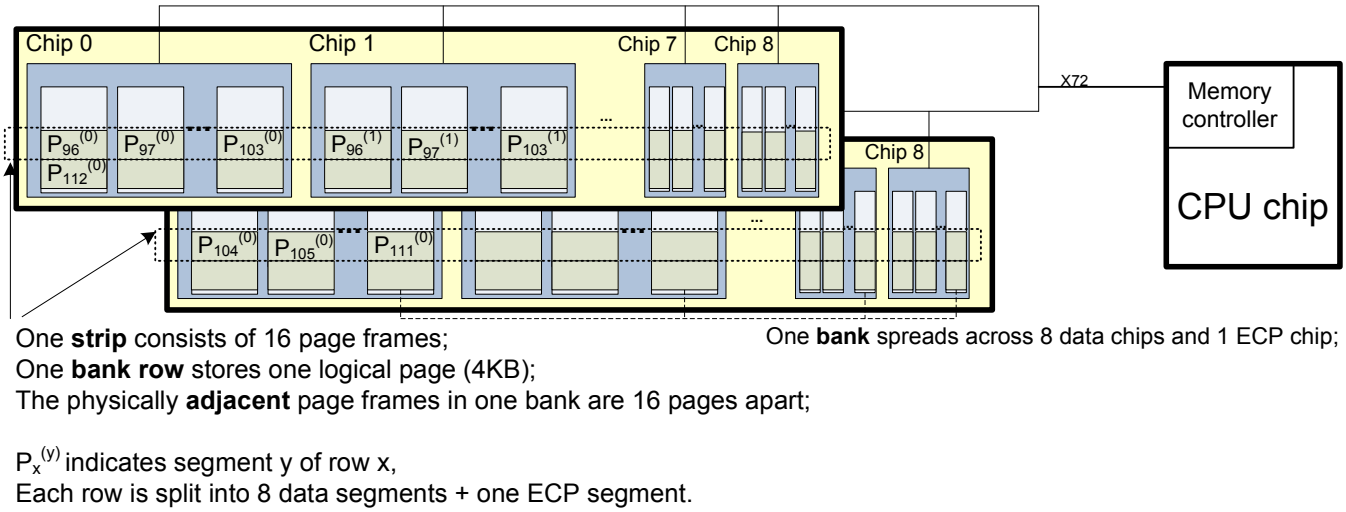


Figure 6. The baseline architecture.

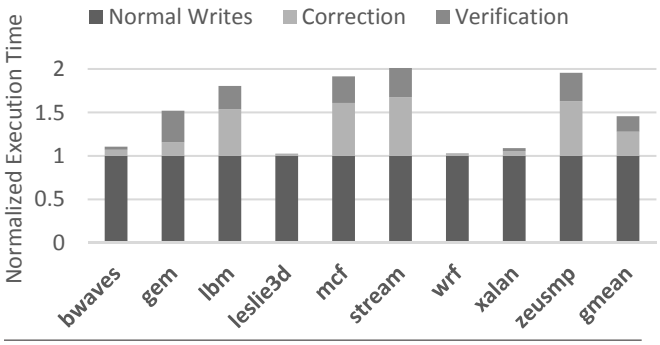


Figure 5. VnC overhead at runtime.

to correct. Thus, writing adjacent lines requires additional VnC processing, resulting in cascading verification and large performance loss.

Figure 5 shows the performance degradation when applying the basic VnC. The experimental setting is in Section 5. From the figure, the overheads for verification and correction are 19% and 28%, respectively. In total, the performance loss is about 47%.

Challenge 2: How to reduce VnC performance impact with low hardware cost?

4. SD-PCM: Enable Reliable Write Operations in Super Dense PCM

4.1 Design Overview

In this paper, we devise SD-PCM to enable reliable writes in super dense PCM (i.e., $4F^2$/cell). In particular, we focus on mitigating WD errors along bit-lines and adopt DIN [10] to minimize WD errors along word-lines. The techniques developed in this paper allow us to choose the densest cell array (Figure 1(a)). As a comparison, the densest cell array of previous PCM designs is shown in Figure 1(c), reflecting 50% memory capacity loss from SD-PCM.

An overview of the baseline architecture of SD-PCM is shown in Figure 6. One DIMM consists of two ranks and each rank has eight banks. At the device level, one bank stores 4096 SLC cells in one row, which spreads across eight PCM data chips. That is, one device row holds the content of one logical page in the OS. There is one additional chip that adopts ECP scheme to defend against hard errors [28].

For discussion simplicity, the rows with the same index from all banks form a *strip*. To maximize parallelism at runtime, a modern OS often maps its physical page frames interleaved across multiple ranks and banks. [17], i.e., one strip consists of 16 consecutive physical pages, and the pages saved in two adjacent rows in one bank are 16 pages apart.

Our design is built on top of the basic Verify and Correct (VnC). Figure 5 illustrates that both verification and correction operations introduce non-negligible performance overhead. Therefore we develop two schemes, LazyCorrection and PreRead, to reduce the overheads, respectively. To further minimize performance impact from VnC and meet the demand of high priority applications, we develop (n:m)-Alloc for achieving good runtime tradeoff between VnC overhead minimization and memory capacity loss.

4.2 LazyCorrection: Reduce Correction Overhead with WD-free ECP

On average, one PCM line write triggers two WD errors in each of its adjacent lines. The hardware cannot start correcting disturbed cells until after the write and the verification steps finish. It is clearly ineffective to prolong the write latency by two RESETs (and further verification) just for a small number of incorrect cells. In this paper, we propose to record error locations and values in the ECP region, and rectify the adjacent lines only if there are more errors than

what ECP can handle. In the design, we prioritize the allocation of ECP entries to hard errors, and handle WD errors using unused ECP entries. In the case if hard errors used up all ECP entries of one PCM line, the WD mitigation for this line would rollback to the basic VnC strategy.

Naively writing ECP region faces the same WD issue, which needs expensive VnC process. To prevent cascading verification and correction, we propose to utilize low density ECP chip (i.e., $8F^2$/cell) for storing ECP pointers (as shown in Figure 7). By buffering WD errors in ECP and utilizing low density ECP chips, we effectively reduce per-write correction overhead and greatly minimize the possibility of cascading verification. The is referred to as LazyCorrection.

LazyCorrection is performed after verification and thus never leaves an adjacent line in erroneous state. Assume we use ECP-`N` where `N` represents the maximal number of errors that ECP can protect, and an adjacent line has accumulated `X` errors (WD errors and/or hard errors) before the current line write. Assume the after-write verification step detects `Y` new errors. The correction step may be skipped if (`X+Y`$\leq$`N`). Otherwise, the hardware tries to correct all (`X+Y`) cells. Some ECP pointers may record hard errors, the correction write can only clear WD errors. The hard errors are still recorded in the ECP region. This correction triggers cascading verification, which reads and verifies its adjacent lines, until no error detected after one verification.

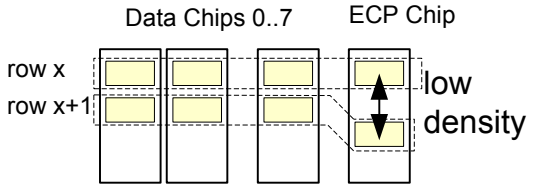


Figure 7. Buffering WD errors in low density ECP region.

When WD errors appear, there is a demand to correct disturbed cells. Without LazyCorrection, each demand is translated to a correction operation. LazyCorrection improves performance by consolidating several correction demands into one correction operation — at the time when there are more WD errors than what ECP can handle. LazyCorrection also consolidates correction demands into normal write. That is, a normal write operation clears the accumulated WD errors in ECP, and thus hides its impact.

A disadvantage using low density ECP chip is that it holds fewer rows comparing to a super dense ECP chip. To ensure ECP protection for every data row, we either need to increase the size of our low density ECP chip, or integrate two ECP chips (of the same size as data chips). In this paper, we adopt the former, which has 100% larger cell array. The latter is also of modest cost. While integrating two ECP chips leads to large cost, it helps to protect 100% more memory capacity. If the memory is built using low capacity data chips, protecting the same memory capacity also requires two ECP chips. In addition, the external data bus is the same (72 bits). For either choice, one memory accesses access only one ECP chip.

4.3 PreRead: Reducing Verification Overhead by Reading Adjacent Lines Early

While LazyCorrection effectively reduces the correction overhead, the performance loss due to verification of adjacent lines is still significant. Comparing to WD-free low density PCM, SD-PCM requires four extra reads operations — two pre-write read operations and two post-write read operations. The former buffers the old data before possible disturbance while the latter checks if the disturbance does happen.

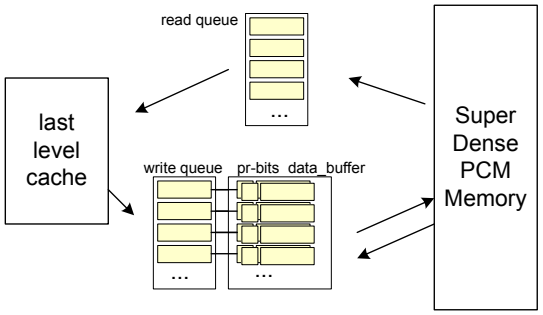


Figure 8. PreRead proactively loads old data of adjacent lines.

To minimize the performance impact of extra read operations, we propose PreRead, which performs the two pre-write reads when the write request is still in the write queue. PreRead is effective because: (i) by default, SD-PCM buffers write requests and flushes those writes when the queue is full, which triggers bursty write. A write request often experiences a long queuing time during which the PreRead operation can be scheduled. (ii) Studies showed that the memory banks in a PCM-based main memory system have a lot of idle periods [23]. Thus, a PreRead operation often has the opportunity to be issued when its associated memory bank is idle, which introduces low performance impact.

Figure 8 shows the hardware enhancement to the write queue to enable PreRead — each entry is enhanced with two PreRead flag bits and two 64B buffers. The flag bits are used to indicate if PreRead operations have been issued for one or both adjacent lines. The return data are buffered in the data buffers for later verification. Note that an adjacent line is 16 physical frames away from the line to be written. Since the hardware uses the physical address directly for PreRead, and may not know its corresponding logical page address, the fetched data is buffered for data verification only. PreRead is not prefetching as the latter needs to forward the fetched data to the upper level caches.

Pre-write reads are not as critical as $real$ read requests, i.e., those from the OS or user applications. They have lower priority for scheduling. For example, when combining PreRead and write cancellation [22], a real read request may

cancel both write and pre-write read requests. This helps to minimize VnC overhead on application performance.

PreRead needs to handle the case when two consecutive lines reside in the buffer simultaneously. For example, buffering the write to line i needs to PreRead both line i-x and line i+x. Here we assume two adjacent PCM lines are x lines away due to interleaving memory layout. If a new write request modifies line i+x and thus needs to PreRead line i, instead of naively reading the data from the memory, we forward the up-to-date data of line i from the buffer directly. This is necessary because, by the time we write line i+x, line i should have already been updated and the memory holds the new data.

4.4 (N:M)-Alloc: A WD-aware Page Allocator for Flexible VnC Overhead Minimization

SD-PCM effectively reduces the average VnC overhead through LazyCorrection and PreRead. However, it still imposes non-negligible performance degradation on memory write-intensive applications. This could become a concern if these applications have high priority and need to finish within tight time constraints. In this section, we present (n:m)-Alloc ($0<n\leq m$), a WD-aware page allocator for minimizing VnC overhead on write-intensive applications.

(N:M)-Alloc uses n out of m consecutive device strips within a block. It is proposed based on the observation that, when writing a PCM line, it is necessary to verify and correct both of its adjacent lines only if these lines store useful data. If the OS can proactively mark one or both adjacent lines as *no-use*, i.e., not to be allocated to any process, the VnC overhead can be reduced or even eliminated, which greatly speeds up the write operation. For example, since (1:2)-Alloc uses every other device strip, there is no need to perform VnC at runtime. When using (2:3)-Alloc, each PCM line has one adjacent line that stores data, and one that does not. A write operation needs to verify and correct only one adjacent line, incurring less overhead.

The design goal of (N:M)-Alloc is to achieve a trade-off between performance and capacity. While it disables a subset of rows, it gains the flexibility to match the performance demands of various applications.

We mark strips as *no-use* within each 64MB independently. As we will elaborate next, the OS performs (n:m) allocation on blocks whose sizes are $2^i \times$64MB ($i\geq 0$). A (2:3) allocator marks the 2nd strip of each 3-strip group. A 3-strip group may span across a 32MB block boundary, but never across any 64MB block boundary. Given a larger block, e.g., a 128MB block, we treat it as two 64MB blocks in marking strips.

(n:m) page allocation. We integrate (n:m) allocation with the buddy-system based page allocation in modern OS. The OS performs (1:1) allocation by default, i.e., no marking, and (n:m) allocation ($n\neq m$) only if it receives explicit requests from the kernel or user applications. The OS manages the page allocation for each (n:m) allocator independently and maintains a separate free-block-list-array. A new field — (n:m) allocator tag, is added to the page table to indicate from which allocator the page is allocated. This tag is then loaded to the TLB and passed to the memory controller.

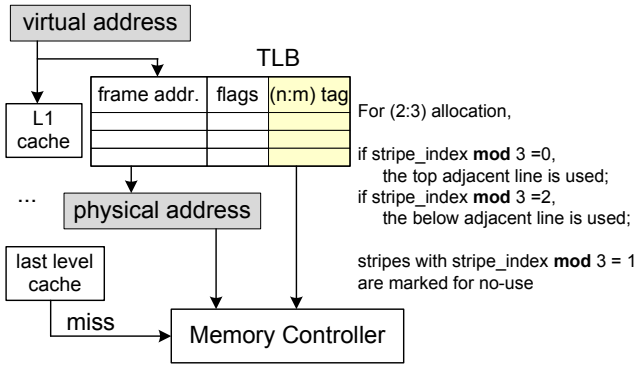


Figure 9. Enhancing TLB with (n:m) allocator tag.

Given a PCM line to be written, the memory controller uses the allocator tag to determine if its adjacent lines are used. Additional verification and correction commands are sent to PCM only if one or both adjacent lines are not marked for *no-use*. Figure 9 illustrates the enhancement to the page table and the TLB entry. After having the physical memory address, the memory controller first computes its strip index within its 64MB block, and then determines which of its adjacent lines need to be verified.

To ensure reliability, if a PCM line belongs to the first strip of its 64MB block, the hardware always verifies its top adjacent line; if a line belongs to the last strip of its 64MB block, its below adjacent line is always verified.

Integrating (n:m)-Alloc with buddy-system based page allocation. In the baseline buddy-system, the OS maintains lists of blocks that are power of two number of pages, e.g., a 2^3-page-sized block list (abbrev. 2^3-list). Given a memory request asking for 16 pages, the OS gets one block from the 2^4-list to service the request. If there is no block on the 2^4-list but one on the 2^5-list. The OS splits the 2^5-page block, returns a 16-page block to the requester, and links the other 16-page block to 2^4-list.

Our WD-aware buddy page allocator maintains one free-block-list-array Free-(n:m) for each (n:m) allocator. (1:1)-Alloc is the same as the baseline allocator as it allocates all strips within a region. Free-(1:1) is the only free list array if no other (n:m) allocator ($n\neq m$) is requested. If there is a request for (n:m) allocation and Free-(n:m) is empty, the OS gets one 64MB block (or a power two number of 64MB block if the requested size is larger) from Free-(1:1) and places it in the appropriate entry in Free-(n:m).

Using (1:2) allocation as an example, we illustrate how to perform page allocation. Assume there is a request asking for 32 pages (128KB). Since it is bigger than a device strip (16 pages), we adjust the requested size and allocate a block

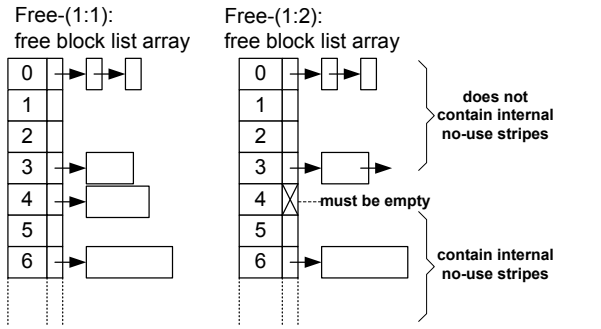


Figure 10. Enabling (n:m)-Alloc in the buddy system for page allocation.

with 64 page frames. The OS will then associate logical pages 0 to 31 to physical frames 0 to 15, and 32 to 47, respectively. The *no-use* strips become *internal fragments*. If there is a request asking for 8 pages (32KB). We service the request directly if there are entries in 2^3-list. If not, we find a block from 2^5-list. All (n:m) allocators with n$\neq$m do not use 2^4-list because requests asking for 16 pages always have their sizes adjusted to 32 pages. When splitting a block that is 16-page or bigger, the OS only uses/links the sub-blocks that are not marked as *no-use*. In the example, the 32-page block is split into two 16-page blocks. The first block is returned to satisfy the 8-page request while the second is marked as *no-use*. The first block is then split to two 8-page blocks with the first one returned to service the original request, and the second linked to 2^3-list. The 16-page *no-use* block becomes *external fragment*.

When freeing a block, the corresponding *no-use* blocks are also reclaimed. In particular, freeing a 16-page block in (1:2)-Alloc automatically forms a 32-page block after reclaiming its *no-use* buddy. An (n:m) allocator (n$\neq$m) can return its 64MB or bigger blocks to (1:1)-Alloc to form larger blocks and reduce fragmentation.

DMA support. While a paging system may freely associate logical pages to physical frames, some operations may demand consecutive page frames, e.g., a DMA operation uses physical addresses to determine the source and target of the moved data, and thus demands for consecutive physical pages. To enable WD-aware DMA operation, we communicate the allocator tag to the DMA controller. For simplicity, we only allow (1:1) or (1:2) allocation choices. For (1:1) allocation, the DMA controller performs the same as the baseline, For (1:2) allocation, the DMA controller skips every other strip automatically.

5. Experimental Methodology

5.1 Baseline Configuration

To evaluate our proposed schemes, we conducted simulation using an 8-core in-order CPU with PCM based main memory. We used an in-house simulator that models the entire memory hierarchy including L1, L2 and DRAM last level cache, PCM based main memory, and memory controller. By default, memory controller buffers incoming write requests and flushes them when write queue is full, which triggers bursty write. Bursty write blocks incoming read requests until it finishes. All timing constraints, memory bank conflict and DDR scheduling constraints are included in simulation as well. Due to the power constraint and the limited number of write drivers, the hardware supports up to 128 SLC cell writes in parallel. The detail of simulation parameters is summarized in Table 2.

CPU	8-core single issue in-order CMP, 4GHz
L1	private, I/D separate, 32KB per core, 64B line
L2	private, 2MB per core, 4-way LRU, 64B line, write back
L3	DRAM cache, private, 32MB per core, 8-way LRU, 64B line, write back, 50ns (200-cycle) hit
Main Memory	8GB, 1 channel, 2 ranks, 8 banks per rank 32-entry write queue per bank
PCM latency	Read: 100ns(400 cycles) Write (SET): 200ns (800 cycles) Write (RESET): 100ns (400 cycles) SLC: 128-bit parallel write [8]

Table 2. Baseline Configuration

5.2 Simulated Workloads

We selected programs from SPEC2006 [2] and STREAM [3] benchmark suites to construct simulated workloads. Table 3 lists the RPKI/WPKI (read/write per thousand instructions) of each program. We used the PIN tool [1] to capture and filter 10 million references to main memory (which normally correspond to several billion instructions of execution time) after skipping application initialization phase and after warming up caches. In the experiment, each core runs one copy of these applications, forming multi-programming workloads running in different virtual address spaces.

Suite	Benchmark	RPKI	WPKI
SPEC2006	bwaves	17.45	0.47
	gemsFDTD	9.62	6.67
	lbm	14.59	7.29
	leslie3d	2.39	0.04
	mcf	22.38	20.47
	wrf	0.14	0.02
	xalan	0.13	0.13
	zeusmp	4.11	3.36
SREAM	copy,scale,add and triad	2.32	2.32

Table 3. Simulated Applications

We adopted the performance metric *Speedup* as

$$Speedup = CPI_{base}/CPI_{tech}$$

from [22] and [10], where CPI_{base} and CPI_{tech} are the cycles of execution from the baseline setting and the setting with different schemes described in 5.3.

5.3 Compared Schemes

We implemented and compared the following schemes.

- `DIN` — This scheme implements DIN-enhanced SLC PCM ($8F^2$/cell, as shown in Figure 1(c)). DIN [10] helps to mitigate WD errors along word-lines. 4F inter-cell space helps to achieve WD-free along bit-lines.
- `baseline` — This scheme implements the basic verify-n-correct (VnC) on super dense PCM ($4F^2$/cell, Figure 1(a)). It often runs into cascading verification due to WD errors along bit-lines.
- `LazyC` — This scheme implements $LazyCorrection$ on top of `baseline`. By default, we used 6 ECP entries for each PCM line (64B).
- `PreRead` — This scheme implements $PreRead$ on top of `baseline` for super dense PCM ($4F^2$).
- `(n:m)-Alloc` — This scheme implements (n:m) allocation on top of `baseline`. For simplicity, we assume one application uses one (n:m)-Alloc for all of its memory demands. In a real system, an application may demand (n:m) allocation (n≠m) only for performance-critical data structures.

6. Results

6.1 PCM Capacity Gain

SD-PCM enables the construction of super dense PCM with $4F^2$/cell while `DIN` uses $8F^2$/cell. The memory capacity improvement for the same sized cell array is 100%. In SD-PCM, `LazyC` needs low density ECP cell array. Therefore, the cell arraies of data chips are of the same size while the cell array of the ECP chip is two times bigger. In `DIN`, both data and ECP cell arrays are of the same size. For fair comparison, we analytically increase the cell array size in `DIN` such that the total sizes of all cell arraies are the same in two settings. This gives 4GB for SD-PCM and 2.22GB for `DIN`, i.e., SD-PCM achieves 80% ($= (4 - 2.22)/2.22$) memory capacity improvement for the cell arrays.

If using chips of the same size, to construct 4GB memory, `DIN` needs 16+2 chips while SD-PCM needs 8+2 chips. This reflects approximately 38% chip size reduction. If using bigger chips for low dense PCM, `DIN` needs 8+1 big chips while SD-PCM need 8 small chips and 1 big chip. The small chip is 23% smaller (as cell array accounts for 46.6% chip area [8]). This reflects approximately 20% ($= (0.77 \times 8 + 1)/(8 + 1)$) chip size reduction.

6.2 Design Overhead Analysis

SD-PCM introduces low hardware cost. LazyCorrection is integrated with existing ECP design. While it asks for low density ECP chips, the external bus width keeps the same, which incurs low hardware overhead. PreRead needs to add two flag bits and two 64B buffers for each entry in the write queue. This accounts for $(64B + 2b) \times 32 \times 2 = 4KB$ for a 32-entry write queue. Compared to the original write buffer size, 2KB, this increment is moderate. While the associativity of write buffer remains the same, search complexity is almost equal to baseline design. In our experiments, we found that its impact on performance is negligible. (n:m)-Alloc adds one 4-bit flag to support 16 different allocators. This accounts for bigger overhead. However, page table usually has many bits reserved. The overhead is insignificant.

6.3 System Performance

Figure 11 compares the performance of different schemes. The results are normalized to `baseline`, i.e., the basic VnC scheme. From the figure, `baseline` has on average 31% performance degradation from `DIN`. This is because of the extra verification and correction operations performed for each write operation.

`LazyC` achieves 21% performance improvement over `baseline`. This is because `LazyC` effectively reduces the correction overhead. We observed 30% and 31% performance improvement when combining `LazyC` with `PreRead` and (2:3)-Alloc, respectively. For memory intensive benchmarks, such as *lbm*, *mcf*, *zeusmp*, `LazyC+(2:3)-Alloc` achieves more benefits over `LazyC+PreRead`. This is because the memory banks are less idle when running these benchmarks — there is less chance to fetch adjacent lines early. On average, `PreRead` and `(2:3)-Alloc` achieve comparable results, when combining with `LazyC` individually.

Combining `LazyC`, `PreRead` and `(2:3)-Alloc` achieves further improvement. On average, it achieves 37% improvement over `baseline`, which is around 5% degradation from `DIN`. Using the most aggressive allocator `(1:2)-Alloc` can effectively eliminate VnC overhead. It does not need to be combined with `LazyC` or `PreRead` as verification and correction operations are not needed.

6.4 Sensitivity to ECP Entries

`LazyC` buffers WD errors in ECP region. Since WD errors accumulate fast, the more that ECP can protect, the fewer corrections `LazyC` shall trigger. The tradeoff is that storing more ECP pointers need more storage space.

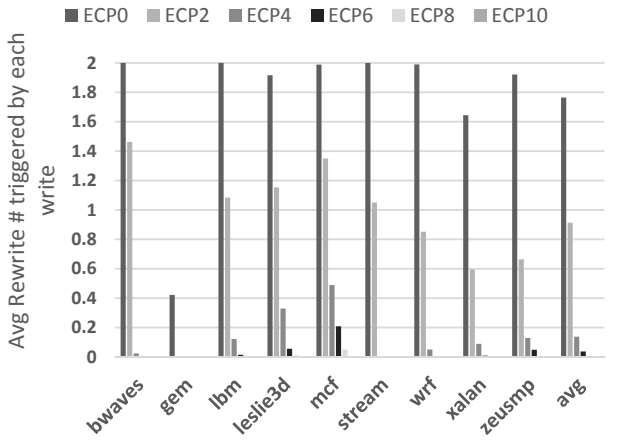


Figure 12. ECP entry number affects correction operations.

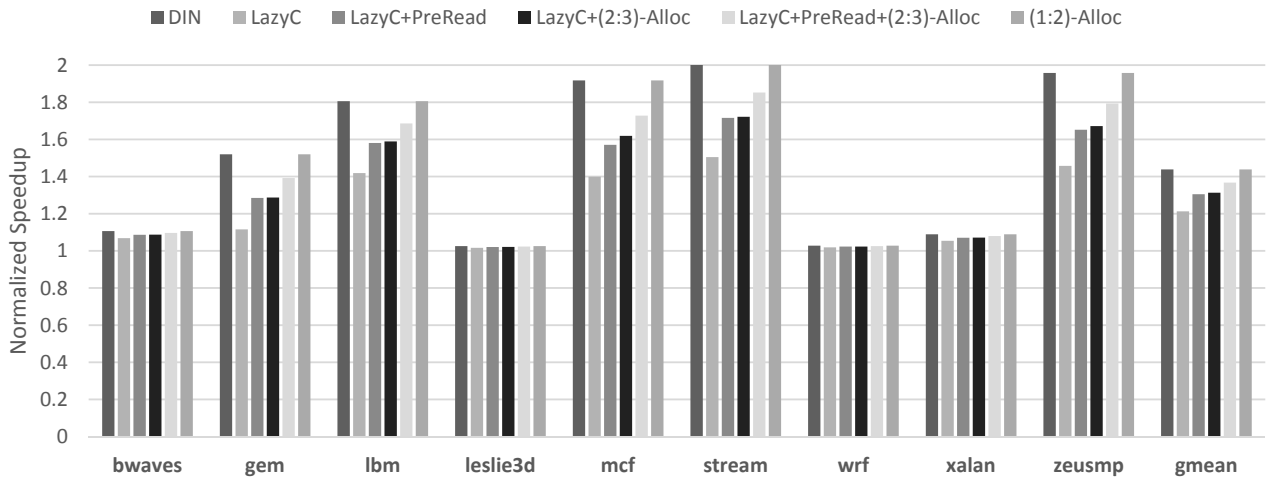


Figure 11. System performance under different schemes (normalized to `baseline`, bigger is better).

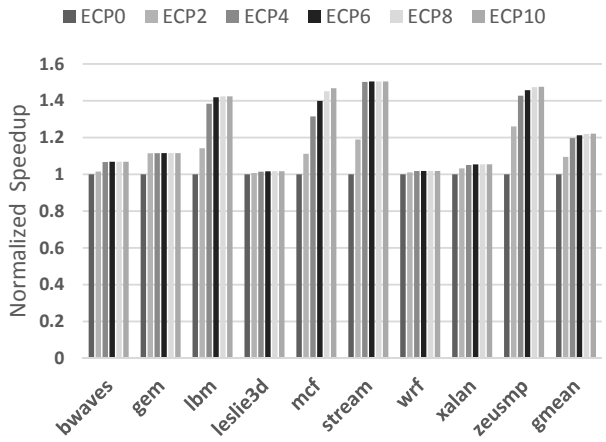


Figure 13. ECP entry number affects system performance.

Figure 12 reports the number of correction operations when having different ECP entries. ECP-0 uses no ECP and thus is the same as `baseline`. From the figure, ECP-0 triggers 1.8 corrections on average, indicating that a write request in `baseline` almost always need to correct both of its adjacent lines. The benchmark $gemsFDTD$ changes less bits per write, leading to fewer WD errors, and fewer correction operations.

When ECP entries grow, the number of correction operations per write decreases significantly. From the figure, ECP-4 triggers on average only 0.14 writes. Only for one highly memory-intensive benchmark *mcf*, ECP-8 shows observable result (0.04 correction per write). For most benchmarks, the default setting ECP-6 is sufficient.

Figure 13 compares performance when having different ECP entries. The results are normalized to `baseline`. Increasing ECP entry number from 0 to 6 helps to achieve 21% improvement. Further increasing the entry number shows negligible improvement.

Lifetime impact. ECP was designed for protecting hard errors. In SD-PCM, `LazyC` uses ECP entries if hard errors do not use up all ECP entries. If there are two hard errors, `LazyC` can only protect up to four WD errors when using ECP-6. Clearly, the number of hard errors increase as ECP chip approaches its lifetime. Figure 14 plots the performance degradation at different times throughout the lifetime of PCM DIMM. Here we ignore the impact of shrinking memory spaces, such as more page faults in the system. From the figure, we observed 0.2% performance degradation when the DIMM is approaching the lifetime limit. Clearly, this is negligible as an aging DIMM has smaller memory capacity, which often has a much larger impact on overall system performance.

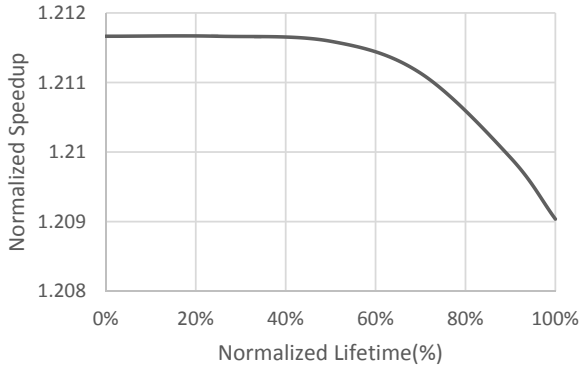


Figure 14. ECP entry number affects system performance.

6.5 Sensitivity to Write Queue Size

In `PreRead`, the memory controller needs to scan the write queue to issue pre-write read operations. Increasing the queue size helps to improve the opportunity of finding an appropriate write whose requested bank is currently idle. Figure 15 summarizes the system performance when having different numbers of entries in the write queue, ranging from 8 to 64.

Increasing the queue size is beneficial only for memory intensive workloads such as *mcf* and *gemsFDTD*. On average, a write queue with 32 entries per bank is sufficient — in `LazyC+PreRead`, it helps to minimize the performance degradation within 10% from `DIN`.

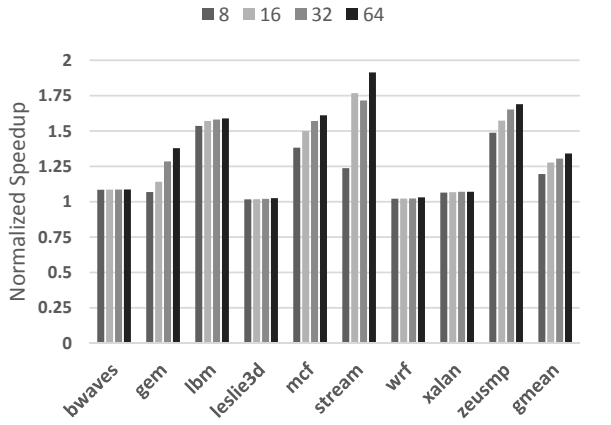


Figure 15. Using different queue sizes in `LazyC+PreRead`.

6.6 Sensitivity to n:m Ratio in (N:M)-Alloc

Figure 16 plots the performance impact using different (n:m allocators. We ignore the impact due to memory capacit loss in (n:m) allocation. Since the (n:m) ratio determines th percentage of strips used for storing user data, the larger th ratio is, the less the memory is wasted, the fewer strips th OS shall mark, and thus the larger performance degradatio we expect.

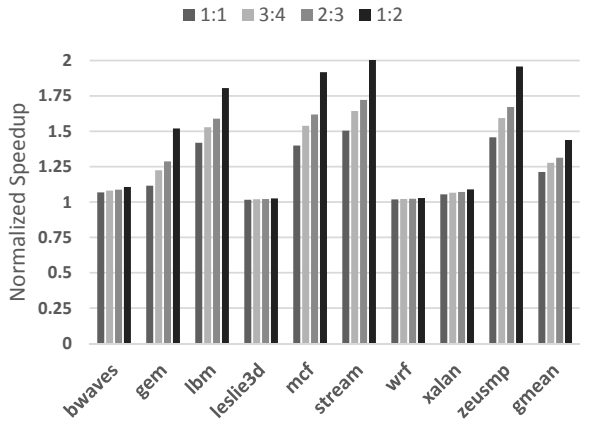


Figure 16. Performance when using different (n:m) alloca tors.

From the figure, 1:2 ratio achieves no performance degra dation as it effectively eliminates VnC by allocating one strip as thermal band between any two strips that store data. Fron. ratio 3:4 to 2:3 until 1:2, performance increases monotonically. This helps to achieve predictable performance at runtime, and adjust the allocation to the need of high priority applications.

6.7 Lifetime Comparison

Figure 17 and 18 present the normalized lifetime degradation on data and ECP chip, respectively. The lifetime degradation of data chip is mainly due to correction accesses. While WD brings up reliability concern, the absolute error number is still low when comparing to the number of cell changes in a normal line write. The figure shows that data chips expects 0.04% lifetime degradation.

Since each WD error translates to writing 10 bits (=9-bit address and 1-bit value) in ECP chip, the lifetime degradation of ECP chip is more significant. We observed an average of 8% degradation. However, ECP was used for protecting hard errors. The cell change rate in ECP chip is low. Without considering WD, ECP chip exhibits 10x longer lifetime than data chip — it is the data chip that determines the DIMM lifetime. Therefore, the lifetime is about the same in SD-PCM.

An alternative ECP design allows intra-row wear leveling across data and ECP chips. This improves DIMM lifetime about 12.5% [28]. Due to low-density ECP design, SD-PCM does not support this design alternative, which leads to a potential of 12.5% lifetime degradation. Note SD-PCM supports intra-row wear leveling among multiple data chips.

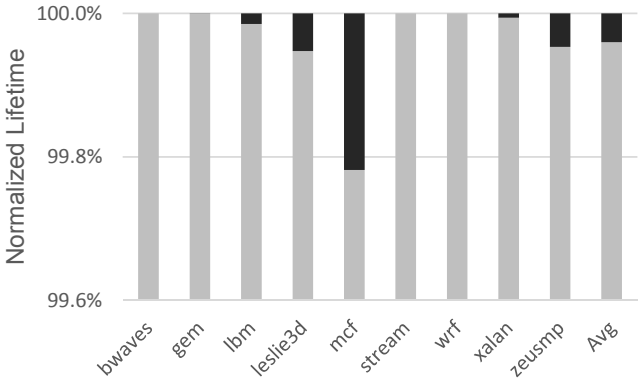


Figure 17. Normalized lifetime degradation on data chips.

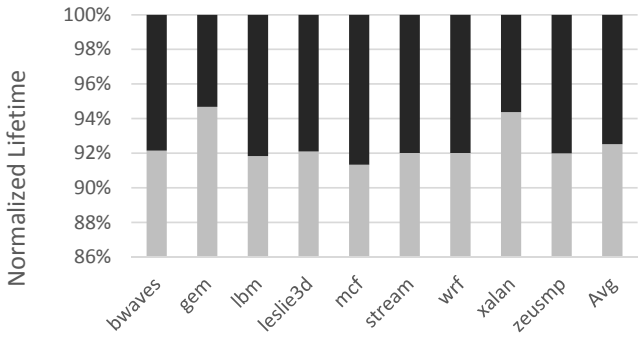


Figure 18. Normalized lifetime degradation on ECP chip.

6.8 Integration with Write Cancellation

VnC effectively prolongs write latency as it performs extra verification and correction operations for many writes. We

next studied the effectiveness of combining SD-PCM with write cancellation, a scheme for reducing the impact of long PCM write latency on system performance [22].

Figure 19 summarizes the normalized performance under different schemes. Write cancellation WC improves VnC but not significantly. This is because WD errors appear more frequently, which significantly prolong write operations — a write operation becomes 2 pre-write reads, 1 write, 2 post-write read and verify, plus 2 correction write operations(RESET), and additional verifications for correction writes not further affecting their adjacent lines. Canceling writes in super dense PCM is not desirable as repeated write operations tend to introduce more WD errors on adjacent lines, which increases the possibility of cascading verification.

By combining WC with LazyC, we improve the performance from 21% to 31%, indicating that WC and LazyC explores different benefits. WC improves performance by prioritize read operations while LazyC focuses on minimizing correction overhead. We did not integrate PreRead due to different scheduling choices in handling write operations.

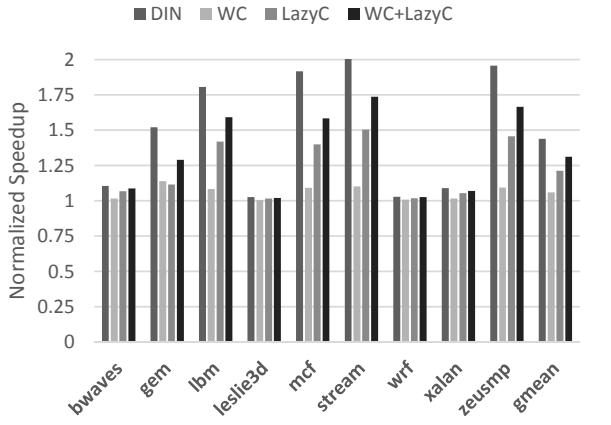


Figure 19. Integrating LazyC with Write Cancellation.

7. Related Work

Phase change memory has become one of the most promising alternatives to traditional DRAM to implement main memory system. Studies have been done on reliability issues in PCM, but mainly focusing on hard errors and resistant drift based soft errors.

PCM suffers from short cell endurance, which becomes even worse due to process variations. PCM chip lifetime can be greatly prolonged by schemes that reduces the number of cell change per write, such as differential-write [35] and flip-n-write [7]. By creating randomized mapping between physical and device addresses, Start-gap [20] and security-refresh [29] spreads writes evenly to all PCM lines. ECP [28], Safer [30], Free-p [33], and PAYG [19] address cell endurances under process variations.

Resistance drift [11] introduces drift-based "soft" errors in MLC PCM. The drift-based error can be mitigated through encoding [34] or periodical scrubbing over the physical memory space [5]. A recent work [31] utilized three out of four states of 2-bit MLC to tolerate the resistance drifting of the most drifting vulnerable intermediate resistance state.

For write disturbance in PCM, a recent work [10] shrinks the thermal band along word-lines and presents thermal isolation data encoding scheme to ensure write reliability.

8. Conclusions

WD in PCM is becoming a major obstacle for scaling PCM [4, 13]. Existing approaches such as VnC and allocating large inter-cell space either introduce large performance overhead or greatly reduce memory density.

In this paper, we propose SD-PCM that consists of three schemes — LazyCorrection, PreRead, and (n:m)-Alloc. The first two target at reducing correction and verification overheads in VnC, respectively. (n:m)-alloc targets at achieving good tradeoff between VnC overhead minimization and memory capacity loss. These schemes can be adopted all together or independently. For example, given a 5% performance degradation constraint, we may meet it by either adopting the first two schemes or adopting (n:m)-alloc with proper n and m. The latter disables a subset of rows while the former does not. Combining all three techniques together helps to achieve the best trade off between capacity loss and performance degradation, in addition, it helps to minimize capacity loss with given performance loss constraint, and vice versa. Our experimental results showed that SD-PCM achieves about 80% capacity improvement in memory cell arrays, while incurs 0-10% performance degradation over the state-of-the-art design, depending on the (n:m) allocator to choose at runtime.

Acknowledgments

This work is supported in part by NSF grants 1422331 and 1012070. The authors thank all the anonymous reviewers for their advice and comments.

References

[1] Pin tool. https://software.intel.com/en-us/articles/pintool.

[2] Spec2006 benchmarks. http://www.spec.org/cpu2006/.

[3] Stream benchmark. http://www.cs.virginia.edu/stream/.

[4] S. J. Ahn, Y. Song, H. Jeong, B. Kim, Y.-S. Kang, D.-H. Ahn, Y. Kwon, S.-W. Nam, G. Jeong, H. Kang, and C. Chung. Reliability perspectives for high density pram manufacturing. In *IEDM*, 2011.

[5] M. Awasthi, M. Shevgoor, K. Sudan, B. R. R. Balasubramonian, and V. Srinivasan. Efficient scrub mechanisms for error-prone emerging memories. In *HPCA*, 2012.

[6] G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, et al. Phase change memory technology. *Jour-*

nal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures, 28(2), 2010.

[7] S. Cho and H. Lee. Flip-n-write: A simple deterministic technique to improve pram writeperformance, energy and endurance. In *MICRO*, 2009.

[8] Y. Choi, I. Song, and M.-H. Park. A 20nm 1.8v 8gb pram with 40mb/s program bandwidth. In *ISSCC*, 2012.

[9] B. Gleixner, F. Pellizzer, and R. Bez. Reliability characterization of phase change memory. In *EPCOS*, 2009.

[10] L. Jiang, Y. Zhang, and J. Yang. Mitigating write disturbance in super dense phase change memory. In *DSN*, 2014.

[11] D. Kang, J. Lee, J. Kong, D. Ha, and et. al. Two-bit cell operation in diode-switch phase change memory cells with 90nm technology. In *VLSI Technology*, 2008.

[12] M. Kang, T. J. Park, Y. W. Kwon, D. Ahn, Y. Kang, H. Jeong, S. Ahn, Y. Song, B. Kim, S. Nam, H.-K. Kang, G. Jeong, and C. Chung. Pram cell technology and characterization in 20nm node size. In *IEDM*, 2011.

[13] B. Kim, Y. Song, D. Ahn, Y. Kang, H. Jeong, D. Ahn, S. Nam, G. Jeong, and C. Chung. Current status and future prospect of phase change memory. In *ASICON*, 2011.

[14] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA*, 2009.

[15] S. Lee, M. Kim, G. S. Do, S. Kim, H. Lee, J. S. Sim, N. G. Park, S. B. Hong, Y. H. Jeon, K. Choi, H. Park, T. Kim, J. Lee, H. Kim, M. R. Choi, S. Lee, Y. Kim, H. J. Kang, J. Kim, H. Kim, Y. S. Son, B. Lee, J.-H. Choi, S. Kim, J. Lee, S. J. Hong, and S.-W. Park. Programming disturbance and cell scaling in pcm: for up to 16nm based $4f^2$ cell. In *VLSIT*, 2010.

[16] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. Raidr: Retention-aware intelligent dram refresh. In *ISCA*, 2012.

[17] H. Park, J. Choi, D. Lee, and S. H. Noh. Regularities considered harmful: Forcing randomness to memory accesses to reduce buffer conflicts for multi-bank, multi-core systems. In *ASPLOS*, 2013.

[18] F. Pellizzer, A. Pirovano, F. Ottogalli, M. Magistretti, M. Scaravaggi, P. Zuliani, M. Tosi, A. Benvenuti, P. Besana, S. Cadeo, T. Marangon, R. Morandi, R. Piva, A. Spandre, R. Zonca, A. Modelli, E. Varesi, T. Lowrey, A. Lacaita, G. Casagrande, P. Cappelletti, and R. Bez. Novel utrench phase-change memory cell for embedded and stand-alone non-volatile memory applications. In *VLSIT*, 2004.

[19] M. K. Qureshi. Pay-as-you-go: Low-overhead hard-error correction for phase change memories. In *MICRO*, 2011.

[20] M. K. Qureshi, J. Karidis, M. F. andVijayalakshmi Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gapwear leveling. In *MICRO*, 2009.

[21] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memorytechnology. In *ISCA*, 2009.

[22] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano. Improving read performance of phase change memories via write cancellationand write pausing. In *HPCA*, 2010.

[23] M. K. Qureshi, M. Franceschini, and L. L. andAshish Jagmohan. Preset: Improving read write performance of phase change memoriesby exploiting asymmetry in write times. In *ISCA*, 2012.

[24] S. Raoux, G. W. Burr, M. J. Breitwisch, C. Rettner, Y.-C. Chen, R. M. Shelby, M. S. D. Krebs, S.-H. Chen, H.-L. Lung, and C. Lam. Phase-change random access memory: A scalable technology. *IBM J. RES. & DEV.*, 2008.

[25] U. Russo, D. Ielmini, A. Redaelli, and A. Lacaita. Intrinsic data retention in nanoscaled phase-change memories - part i: Monte carlo model for crystallization and percolation. *TED*, 53(12), 2006.

[26] U. Russo, D. Ielmini, A. Redaelli, and A. Lacaita. Modeling of programming and read performance in phase-change memories - part i: Cell optimization and scaling. *TED*, 55(2), 2008.

[27] U. Russo, D. Ielmini, A. Redaelli, and A. Lacaita. Modeling of programming and read performance in phase-change memories - part ii: Program disturb and mixed-scaling approach. *TED*, 55(2), 2008.

[28] S. Schechter, G. H. Loh, K. Strauss, and D. Burger. Use ecp, not ecc, for hard failures in resistive memories. In *ISCA*, 2010.

[29] N. H. Seong, D. H. Woo, and H.-H. S. Lee. Security refresh: Prevent malicious wear-out and increase durabilityfor phase-change memory with dynamically randomized address mapping. In *ISCA*, 2010.

[30] 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 *micro*, 2010.

[31] N. H. Seong, S. Yeo, and H.-H. S. Lee. Tri-level-cell pcm: toward an efficient and reliable memory system. In *ISCA*, 2013.

[32] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA*, 2010.

[33] D. H. Yoon, N. Muralimanohar, J. Chang, ParthasarathyRanganathan, N. P. Jouppi, and M. Erez. Free-p: Protecting non-volatile memory against both hard and softerrors. In *HPCA*, 2011.

[34] W. Zhang and T. Li. Helmet: A resistance drift resilient architecture for multi-levelcell phase change memory system. In *DSN*, 2011.

[35] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memorytechnology. In *ISCA*, 2009.