# PreSET: Improving Performance of Phase Change Memories by Exploiting Asymmetry in Write Times

Moinuddin K. Qureshi† Michele M. Franceschini Ashish Jagmohan Luis A. Lastras

†*Georgia Institute of Technology*
*moin@ece.gatech.edu*

*IBM T. J. Watson Research Center, NY*
*{franceschini, ashishja, lastrasl}@us.ibm.com*

## Abstract

*Phase Change Memory (PCM) is a promising technology for building future main memory systems. A prominent characteristic of PCM is that it has write latency much higher than read latency. Servicing such slow writes causes significant contention for read requests. For our baseline PCM system, the slow writes increase the effective read latency by almost 2X, causing significant performance degradation.*

*This paper alleviates the problem of slow writes by exploiting the fundamental property of PCM devices that writes are slow only in one direction (SET operation) and are almost as fast as reads in the other direction (RESET operation). Therefore, a write operation to a line in which all memory cells have been SET prior to the write, will incur much lower latency. We propose* PreSET*, an architectural technique that leverages this property to pro-actively SET all the bits in a given memory line well in advance of the anticipated write to that memory line. Our proposed design initiates a PreSET request for a memory line as soon as that line becomes dirty in the cache, thereby allowing a large window of time for the PreSET operation to complete. Our evaluations show that PreSET is more effective and incurs lower storage overhead than previously proposed write cancellation techniques. We also describe static and dynamic throttling schemes to limit the rate of PreSET operations. Our proposal reduces effective read latency from 982 cycles to 594 cycles and increases system performance by 34%, while improving the energy-delay-product by 25%.*

## 1 Introduction

Dynamic Random Access Memory (DRAM) has been the basic building block for designing main memories over the past four decades. Unfortunately, scaling DRAM to smaller feature sizes has become difficult. Therefore, memory technologies that promise better scalability than DRAM have become attractive for designing future memory systems [7][4][17]. One of the prime contender among emerging technologies is Phase Change Memory (PCM). PCM is a scalable technology that has read latency close to that of DRAM. Unfortunately, PCM is an asymmetric read-write technology where the write latency is much higher compared to the read latency (often about 8x). A higher write latency can be usually be tolerated using buffers and intelligent scheduling if there is sufficient write bandwidth. However, once a write request is scheduled for service to a PCM bank, a subsequent read request for a different line to the same bank waits until the write request gets completed. Thus, the write requests can significantly increase the effective latency for read requests. Read accesses, unlike write accesses, are latency critical and slowing down the read accesses has significant performance impact. The contention-less read latency in our baseline system is configured to be 500 cycles (experimental methodology is described in Section 4). However, because of the contention from the slow writes, the effective read latency almost doubles, reaching 982 cycles. Our baseline employs large number of banks, read priority scheduling, as well as large per-bank write queues. However, the contention from write requests is still significant. To reduce this contention, Qureshi et al. [8] proposed Adaptive Write Cancellation (AWC) policies that can cancel an on-going write in order to service a pending read request. While AWC indeed reduces effective read latency (to 694 cycles, 1.4x of contention-less read latency), it still leaves significant room for performance improvement. Furthermore, AWC relies heavily on having a large number of write queues entries which incurs significant complexity in the memory controller. We want a scheme that obtains almost all of the potential performance improvement without incurring significant hardware overhead or complexity.

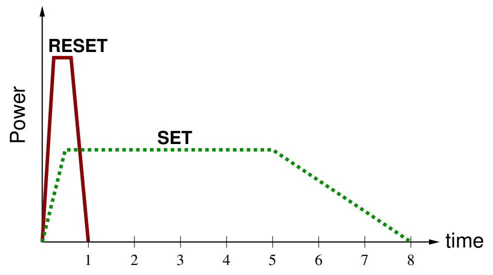

**Figure 1. PCM write characteristics.**

A key insight that enables our solution is that the write latency in PCM devices is data dependent, as shown in Figure 1. To RESET the device a high power short duration pulse is required. The latency for the RESET operation is similar to the read operation [6]. However, the operation to SET the cell, takes a long latency pulse to lower the resistance of the cell, which is typically about 8X longer than the RESET pulse [6]. Given that a memory line contains hundreds of bits, it is highly likely, when writing, that both RESET and SET transitions will occur, hence write latency is determined by the slower of the two operations.

We could reduce the latency of PCM write operations by 8X if the memory writes were constrained to only cause RESET operations. We leverage this key insight and propose *PreSET*, an architectural technique that leverages the write asymmetry of PCM devices to pro-actively SET all the bits in a given memory line well in advance of the anticipated write to that memory line. When a writeback request reaches the memory and the PreSET request has already been serviced for that memory line, writes are completed with much lower latency as only RESET operations are required. Since PreSET is only a hint, it is done off the critical path and is a non-blocking operation. Furthermore, the structures that track status of PreSET requests need to store only address information and not data values, resulting in much simpler and smaller structures.

One of the main questions in architecting a system with PreSET is to decide when to initiate a PreSET request, as performing a PreSET operation for clean lines can result in potential data loss. We use the key insight that when the processor writes to a line in the cache, the corresponding contents of the line in memory becomes stale and can be discarded. Therefore, a PreSET to a memory line can be initiated as soon as a write to the cache line is performed. In fact, the cache line may be written multiple times before it gets written back to memory, but we need only initiate one PreSET operation for that line, as PreSET is not dependent on the content of the line. Our evaluations show that the time between the first time a line is written in the DRAM cache of hybrid memory system and the time it gets evicted is in the range of hundreds of millions of cycles. Therefore, the PreSET operation can be scheduled at anytime during this period, and has a high likelihood of completion before the dirty line is written to memory.

We discuss the extensions to cache architecture and memory system in order to facilitate PreSET. Our evaluations show that PreSET can reduce the effective read latency of our baseline system from 982 cycles to 661 cycles. This compares favorably with adaptive write cancellation (AWC) policies. In fact, combining PreSET and AWC obtains an effective read latency of 594 cycles, and provides an overall performance improvement of 34%. This performance improvement is within 2% of a system that has PCM write latency equal to the read latency. Thus, the proposed techniques alleviate the problem of slow writes in PCM, making the PCM-based system seem like a symmetric read-write system. Furthermore, our analysis shows that if PreSET is implemented, then most of the writes are fast and hence the system does not need provide large write queues, which is otherwise necessary to support. Therefore, PreSET not only improves performance but also helps with reducing the area and complexity of the memory controller.

While PreSET improves performance significantly, it increases the amount of time the memory system spends in performing write operations (either demand writes or PreSET writes). This incurs significant power overheads and reduces system lifetime. To limit the overhead of PreSET while retaining the performance benefits, we propose throttling schemes that regulate the number of PreSET operations. We describe both static and dynamic schemes for PreSET Throttling. The proposed schemes have performance improvement similar to unconstrained version PreSET+AWC but significantly reduces power overhead and lifetime degradation. These schemes improve the energy-delay product of the system by more than 25% and provide a system lifetime of 8+ years on average (5.6 years minimum).

## 2 Background and Motivation

PCM is a non-volatile memory that exploits the ability of chalcogenide glass [16][15] to switch the material between two states, amorphous and polycrystalline. The amorphous phase has high resistance and polycrystalline phase has low resistance. The state of a PCM device is changed by heating. Different heat-time profiles are used to switch from one phase to another. To RESET the device (Figure 2A), a high power pulse of short duration is required. This electrical pulse first raises the temperature of the PCM material above its melting point, typically in excess of 600 C, and is then quickly terminated. The small region of melted material subsequently cools extremely quickly as a result of thermal conduction into the surroundings. This extremely rapid cooling process locks the PCM material into an amorphous state. The small dimensions of typical PCM devices results in a small thermal time constant thus allowing RESET pulses with durations of tens of nanoseconds to be effective. Therefore, RESET latency is typically similar to the read latency [6].

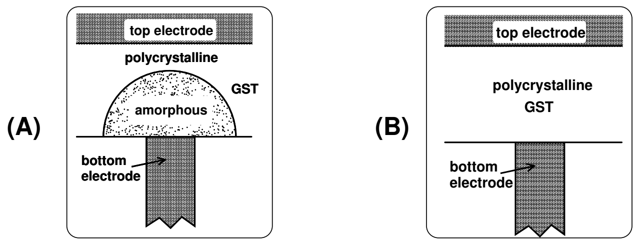

**Figure 2. State of PCM cell after (A) RESET (B) SET**

To SET a cell (Figure 2B), the amorphous material must be encouraged to crystallize into a polycrystalline state. This can be accomplished by heating the material above its crystallization temperature but below its melting point for a sufficient length of time. Since the rate of crystallization is a strong function of the temperature, given the variability of PCM cells within an array, reliably crystallizing typical PCM cells requires heating pulses that are hundreds of nanoseconds in duration. Therefore, the SET latency is much higher (often 8x or greater) compared to the RESET latency [6]. Given that a memory line contains hundreds of bits, it is highly likely, that both RESET and SET transitions will occur during writing, hence the write latency of PCM array is determined by the slower of the two operations.

As write accesses are not in the critical path, a higher write latency can typically be tolerated using buffers and intelligent scheduling. However, once a write request is scheduled for service to a PCM bank, a subsequent read request for a different line in the same bank waits until the write request gets completed. Thus, writes can increase the effective latency for read requests. Read accesses, unlike write accesses, are latency critical and slowing them can have significant performance impact. To tolerate the contention from slow writes, Qureshi et al. [8] proposed Adaptive Write Cancellation (AWC) policies that can cancel an ongoing write request in order to service pending read request.

Figure 3 shows the average effective read latency (on the left) and performance (on the right) for the baseline PCM system with contention-less read latency of 500 cycles (methodology in Section 4). The write latency is assumed to be 8x the read latency and the baseline uses read priority scheduling. For comparison, configurations where the baseline writes incur same latency as reads

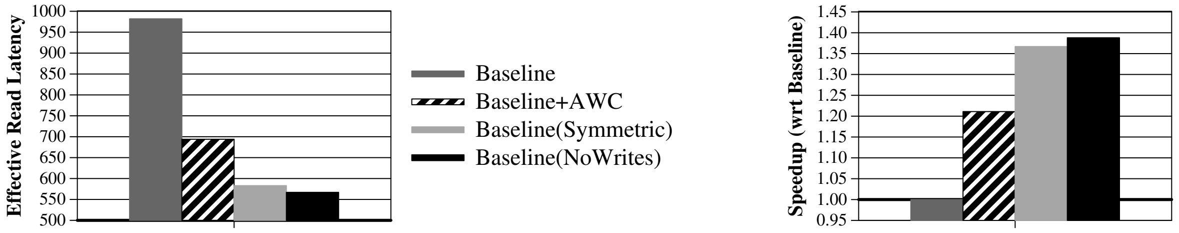

**Figure 3. Effective read latency and relative performance of PCM systems (methodology is in Section 4)**

(Symmetric), and writes consume zero cycle latency (NoWrites) are also shown, in addition to AWC.

The effective read latency for the baseline is 982 cycles, almost doubled compared to the latency without contention. If write latency was identical to read latency (500 cycles) then the effective read latency reduces to 583 cycles, indicating most of the contention is due to writes. If all the writes were removed (NoWrites) then there is a potential to improve performance by 39% (on average). AWC obtains only about half of this potential while relying heavily on large write queues. The goal of this paper is to provide simple extensions to PCM controller that obtain almost all of the potential performance benefit without relying on large write queues. The next section describes our proposal.

## 3 Architecting Memory System for PreSET

The key observation that enables our solution is that the write latency in PCM devices is data dependent, as shown in Figure 1. The latency of RESET is similar to read latency, whereas the latency of SET is much higher (approximately 8X). During a typical memory write, which involves a large number of memory cells, both operations are likely to occur hence the write latency is high. If we were able to constrain memory writes to only RESET operations, we could reduce the latency of PCM write operations by potentially 8X. We leverage this key insight and propose a novel architecture technique called *PreSET* for PCM-based memory systems. Our proposal pro-actively SETs all the bits in a given memory line well in advance of the anticipated write to that memory line. If when a writeback request reaches the memory, a PreSET request has already been serviced for that memory line, the write is completed with much lower latency as only RESET operations are required. This section describes the observation, design support, and policies for enabling effective PreSET in PCM systems.

### 3.1 Initiating PreSET Request

The key question in designing a system with PreSET is when to initiate a PreSET request for a given line. A PreSET request should not be initiated speculatively, as performing a PreSET operation for clean lines can result in potential data loss. Therefore, a PreSET operation must be performed only when it is guaranteed that the current contents of the line will not be needed. We use the insight that when a write is performed to a given cache line, the corresponding contents of the line in memory becomes stale and can be discarded. Therefore, a PreSET to a memory line can be initiated as soon as a write to the cache line is performed. In fact, the cache line may be written to multiple times before eviction, but we need initiate only one PreSET operation for that line, as PreSET is not dependent on the contents of the line.

Figure 4 explains these concepts using lifetime of a cache line from install to eviction. At time $t_0$, the line is installed in the cache. At time $t_1$, the line gets written (possibly by a writeback from the smaller caches) for the first time in the cache. After that it can be written several times, before getting evicted from the cache at time $t_n$, resulting in a writeback to memory. PreSET request can be sent to memory at any time after $t_1$ and must complete before time $t_n$. We define this time period as the *PreSET Window*.

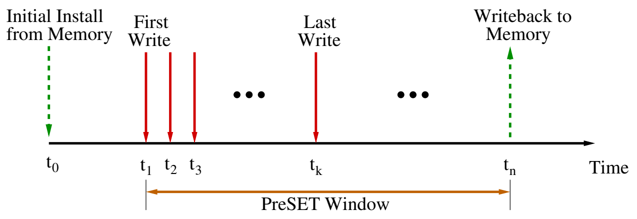

**Figure 4. The window for PreSET is between first write to cache line and writeback to memory.**

### 3.2 Viability of PreSET Window

A PreSET is by construction a long latency operation (4000 cycles in our system), that can get in the way of reads and actual write operations on the memory. It is treated with lowest priority and is serviced only during spare cycles. Therefore, we want the PreSET window to be large, so that the PreSET requests have a high chance of finishing before the corresponding write request arrives at memory.

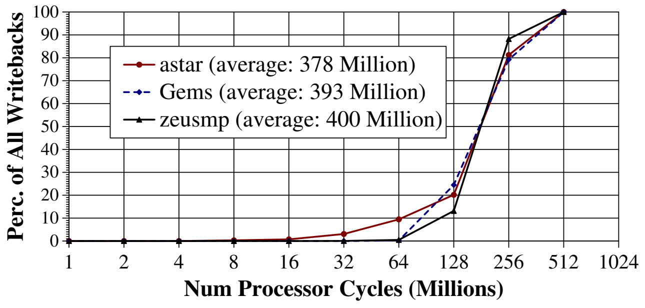

**Figure 5. Cumulative density function of the number of processor cycles between the setting of the dirty bit and the actual eviction of a line from our 32MB DRAM cache.**

While we can use even the write at the first level cache to initiate PreSET, for keeping the design simplified and the processor chip unchanged, we choose to use only the write in the DRAM cache for initiating PreSET. Figure 5 shows the number of cycles

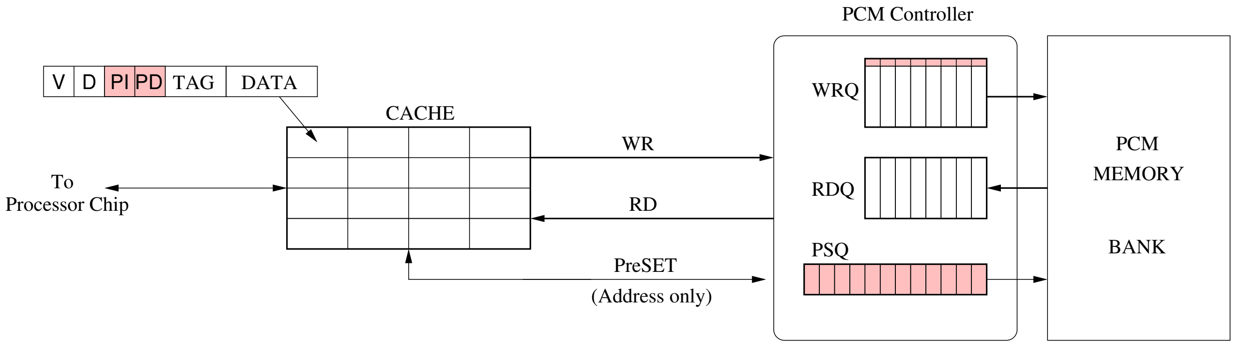

**Figure 6. Architecture extensions to support PreSET (newly added structures/fields are shaded). Figure not to scale.**

between first write and eviction for all writebacks to memory for three typical benchmarks. On average, the PreSET window is in the range of few hundred million cycles. For almost all writes, the window is in the range of few tens of millions of cycles, which far exceeds the time to do a PreSET. Therefore, we expect that in the common case most lines would have PreSET request completed before the corresponding write request appears at memory.

## 3.3 Architecture Support

The cache design and memory system must be extended to facilitate PreSET, as shown in Figure 6. The memory controller has a queue called PreSET queue (PSQ), in addition to the read queue (RDQ) and write queue (WRQ). The PSQ is much simpler than WRQ, in that each PSQ entry stores only address information (3 bytes) whereas each WRQ entry stores data as well address (128+3 = 131 bytes). Therefore, even a PSQ of 128 entry incurs a storage of less than 400 bytes (10X lower than a 32-entry WRQ). Furthermore, the PSQ need not be looked upon each read access, whereas WRQ is always looked upon for each read access. Therefore, PSQ can be designed as a circular buffer (or a set-associative structure) and avoids the complexity of a CAM-structure needed for WRQ.

The tag store entry of the cache is extended to have two more status bits: PreSET Initiated (PI) and PreSET Done (PD). When a write is performed to the cache, the dirty bit (D) is set. A PreSET request is sent to memory, only if PI bit is zero, and the PSQ associated with the bank has empty space. If PI bit is set to 1, subsequent writes to the cache line will not initiate a PreSET request. When the PreSET request completes service, the cache gets notified to set the PD bit. The PD bit avoids looking up the PSQ for writebacks for which PreSET has already been done. When a line gets evicted and if PI=1 and PD=0 (which happens in extremely rare cases), we snoop the PSQ and invalidate the PreSET request to avoid data loss, in case the PreSET operation happens after the demand write.

When a dirty line is evicted from the cache, and the PD bit is set it is inserted with that information in the WRQ (each WRQ is extended to have one status bit $PD$). If the PI bit is set but PD bit is not set, that means the entry is still in PSQ. It is vital that such unfulfilled PreSET request are squashed, otherwise it can result in data loss (if PreSET is performed after the demand write). Therefore, in such scenarios the PSQ entry associated with that line is invalidated. When the memory controller schedules the write operation from WRQ, the PD bit determines whether it should be treated as a normal write (provisioning for both SET and RESET transitions) or a fast write (only RESET transitions will occur).

## 3.4 Interface Support for Bimodal Writes

PreSET operation in itself does not require special support from the device level. It is similar to a normal write where all data bits are zero (or one, depending on the SET to data value mapping). However, we do rely on the interface supporting two types of write latencies: one a slow write (for normal writes or for SET), and another a fast write (for writes that require only RESET).

## 3.5 Scheduling PreSET at Memory Bank

As PreSET request is only a hint, it is serviced off the critical path, during idle cycles at the memory bank. Our evaluations show that the average memory utilization is 30%, indicating lots of opportunities for doing PreSET. A PreSET request is deemed to have a priority lower than both read and writes. So, a request from PSQ is scheduled for service *only* if the RDQ and WRQ of that bank are empty.

A PreSET request can be conceived to be of two flavors: blocking and non-blocking. We evaluated a blocking PreSET that continues even if a read request arrives for that bank. We found that such a blocking PreSET degrades performance compared to the baseline by 5.2%. Therefore, it is vital that PreSET operations are non-blocking and that they get canceled for a later arriving read requests. Implementing non-blocking PreSET requires a memory controller support similar to that for write cancellation [8].

Throughout this paper we will assume that PreSET operations are always non-blocking for read requests. Note that, we do not cancel an ongoing PreSET operation if only a write request arrives at the bank, as the writes can wait in the WRQ for some time without hurting performance, and we found that completing the ongoing PreSET operation in such scenarios improves performance.

## 3.6 Combining PreSET and AWC

Prior work has proposed adaptive write cancellation (AWC) policies for canceling an ongoing write operation in favor of a later arriving read request, depending on how much write service is already completed. While PreSET helps convert most of the writes into short latency writes, some long latency writes may remain. The system can still benefit from canceling the write request. So, we propose and investigate a policy that combines PreSET and AWC, and call it *PreSET+AWC*.

## 4 Experimental Methodology

### 4.1 Configuration

We use an in-house system simulator for our studies. The baseline system is shown in Figure 7 with the parameters from Table 1. We use a simple in-order processor model so that we can evaluate our proposal for several billion instructions. The baseline contains a 256MB write-back DRAM buffer organized as a 32MB per-core private cache. Write requests arrive at main memory only on DRAM cache evictions. The 32GB PCM-based main memory has 4 ranks of 8 banks each. Both read and write requests are at the granularity of cache line, and are serviced by one of the banks based on line address.

Each bank has a separate 8-entry read queue (RDQ) and 32-entry write queue (WRQ) that queues all pending requests. A read request to a line pending in the WRQ is serviced by the WRQ. If both RDQ and WRQ are non-empty then read request is serviced unless the WRQ is $>$ 80% full, in which case a write request is serviced. This ensures that read requests are given priority in the common case, but writes eventually get a chance to get service. For access to PCM-based main memory we assume that the read latency is 125 ns (500 cycles) and writes latency is 1 micro second (4000 cycles). PreSET operation is assumed to take 4000 cycles, and writes with only RESET takes 500 cycles.

**Table 1. Baseline Configuration**

| | |
|---|---|
| System | 8-core single-issue in-order CMP, 4GHz |
| L2 cache (private) | 2MB 4-way, 128B linesize, write-back |
| DRAM cache (private) | 32MB 8-way, 128B linesize, write-back<br>100 cycle access, tag-store in SRAM |
| Main memory | 32GB PCM, 4 ranks of 8-banks each<br>32 entry write-queue per bank<br>Read priority scheduling if WRQ <80% full |
| PCM latency | reads : 125ns (500 cycles)<br>writes: 1 $\mu$s (4000 cycles)<br>write (SET only): 4000 cycles<br>write (RESET only): 500 cycles |

### 4.2 Workloads

We use a representative slice of six benchmarks from the SPEC2006 suite: $astar$, $cactusADM$, $GemsFDTD$, $leslie3d$, $soplex$ and $zeusmp$. These benchmarks were chosen because they have at least 0.6 memory accesses per 1000 instructions out of the 32MB DRAM cache (for workloads that fit in the cache, the proposed scheme neither helps nor hurts performance). We run these benchmarks in a rate mode. We also use six multiprogrammed workloads each containing 2 copies of four benchmarks.

We perform timing simulation till the main memory services 10 million references (this translates to several billion instructions of execution time). Table 2 shows the percentage of cycles a PCM bank is servicing a read (Read Utilization), write (Write Utilization), and Idle on average for the baseline system. The idle periods provide opportunity for performing PreSET operations.

**Table 2. Benchmark Characteristics (Bank utilization are for PCM-based main memory).**

| Name | Description | Bank Utilization (%) | | |
|---|---|---|---|---|
| | | Read | Write | Idle |
| astar_r | 8 copies of astar | 3.4 | 17.3 | 79.3 |
| cactus_r | 8 copies of cactusADM | 6.7 | 15.6 | 77.7 |
| Gems_r | 8 copies of GemsFDTD | 8.6 | 32.6 | 58.8 |
| leslie_r | 8 copies of leslie3d | 9.0 | 25.8 | 65.2 |
| soplex_r | 8 copies of soplex | 11.5 | 17.9 | 70.6 |
| zeusmp_r | 8 copies of zeusmp | 7.4 | 20.1 | 72.5 |
| mix_1 | cactus-leslie-soplex-zeusmp (x2) | 8.9 | 19.3 | 71.8 |
| mix_2 | astar-cactus-leslie-soplex (x2) | 8.3 | 19.0 | 72.7 |
| mix_3 | cactus-Gems-soplex-zeusmp (x2) | 8.9 | 21.1 | 70.0 |
| mix_4 | astar-Gems-soplex-zeusmp (x2) | 8.4 | 21.3 | 70.3 |
| mix_5 | astar-cactus-soplex-zeusmp (x2) | 8.0 | 17.0 | 75.0 |
| mix_6 | astar-cactus-Gems-soplex (x2) | 8.2 | 20.8 | 71.0 |

### 4.3 Figure of Merit

The objective of our proposal is to reduce the latency to service the read requests. If $t_{rdqin}$ is the time at which the read request enters a PCM read queue and $t_{done}$ is the time at which it finishes service by a PCM bank, then we define Effective Read Latency as:

$$\texttt{Effective Read Latency} = (t_{done} - t_{rdqin}) \quad (1)$$

The value of the effective read latency averaged across all read requests is reported as the figure of merit. Also, overall system performance is reported as relative speedup:

$$\texttt{Speedup} = \frac{\texttt{Execution Time Of Baseline}}{\texttt{Execution Time With Proposed Technique}} \quad (2)$$

We found that the performance improvement with this metric is similar (within 1%) to improvement in Weighted Speedup [13].

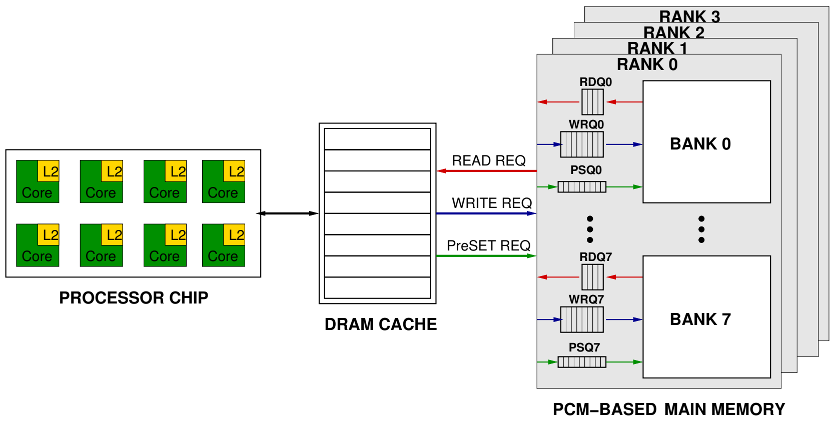

**Figure 7. Simulated Baseline System with support for PreSET (Figure not to scale)**

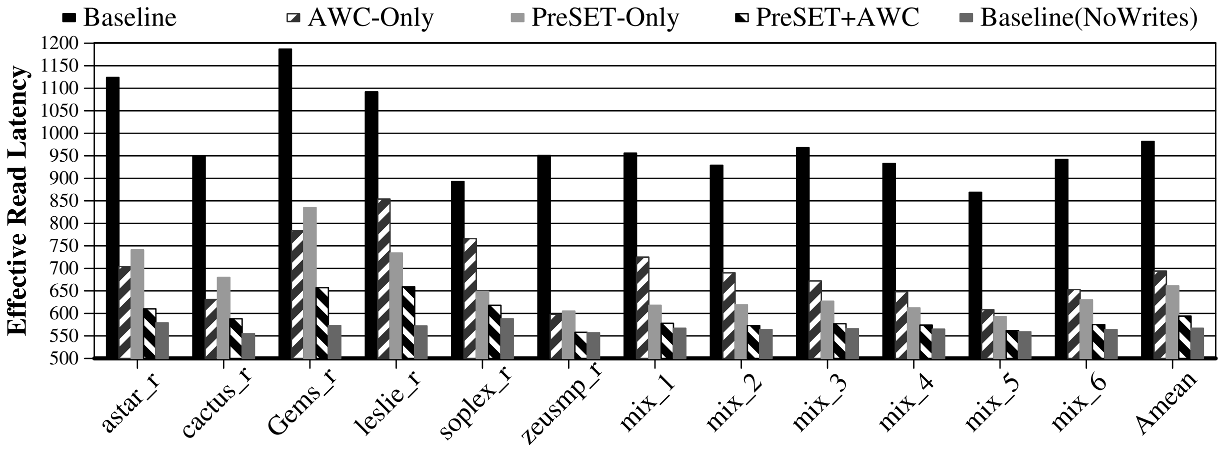

**Figure 8. Comparison of Effective Read Latency of different PCM systems**

## 5 Results and Analysis

### 5.1 Impact on Effective Read Latency

One of the key performance metric for a memory system is the latency to complete a read operation. Figure 8 compares the effective read latency of five systems: Baseline, Adaptive Write Cancellation (AWC), PreSET, PreSET+AWC, and a Baseline system from which all writes are removed (NoWrites). The bar labeled *Amean* represents the average over all workloads. The system NoWrites represents an upper bound on possible performance improvement with techniques that try to reduce contention of writes on reads. For PreSET and PreSET+AWC we assume a 128-entry PSQ ($<$ 400 bytes).

For all workloads, the contention from write requests is significant.[1] For 10 out of 12 workloads, the effective read latency is more than 900 cycles, much higher than contention-less latency of 500 cycles. The average latency for the baseline is 982 cycles. This reduces to 567 cycles, if all writes were removed from the baseline, indicating that most of the latency increase is due to contention from write accesses. AWC reduces effective latency to 692 cycles (1.4X of contention-less read latency, which is similar to that reported by Qureshi et al. in [8]). However, AWC still leaves significant room for improvement. PreSET is more effective than AWC, reducing the average latency to 660 cycles. The combination of PreSET+AWC is highly effective, and obtains an average of 594 cycles, very close to the system with NoWrites. Thus, our proposal alleviates the problem of contention from writes on reads.

[1]The low memory utilization of Table 2 and high read latency of Figure 8 may seem inconsistent with each other to a casual reader. This behavior happens because writes are 8x longer than reads and can be explained with a simple model. If memory is busy with writes 25% of the time, then there is 25% chance that a randomly arriving read will contend with a write. Each such episode will cause the read to wait on average for [8x (write latency)]/2=4X. So, the average contention for read would be 0.25*4X=1X, i.e. effective latency of approximately 1000 cycles for raw latency of 500 cycles (similar to Figure 8).

### 5.2 Impact on System Performance

Figure 9 shows the normalized speedup of AWC, PreSET, PreSET+AWC, and baseline with NoWrites. The bar labeled *Gmean* represents the geometric mean over all workloads. For all workloads, the improvement in read latency translates into significant performance improvement. On average, there is a potential improvement of 38.7% with NoWrites. AWC gets only 21% of this potential, while PreSET obtains 27.8%. With PreSET+AWC, the performance improvement is 34.7%, which is within 4% of the upper-bound NoWrites system, indicating very little room for improvements from further optimizations.

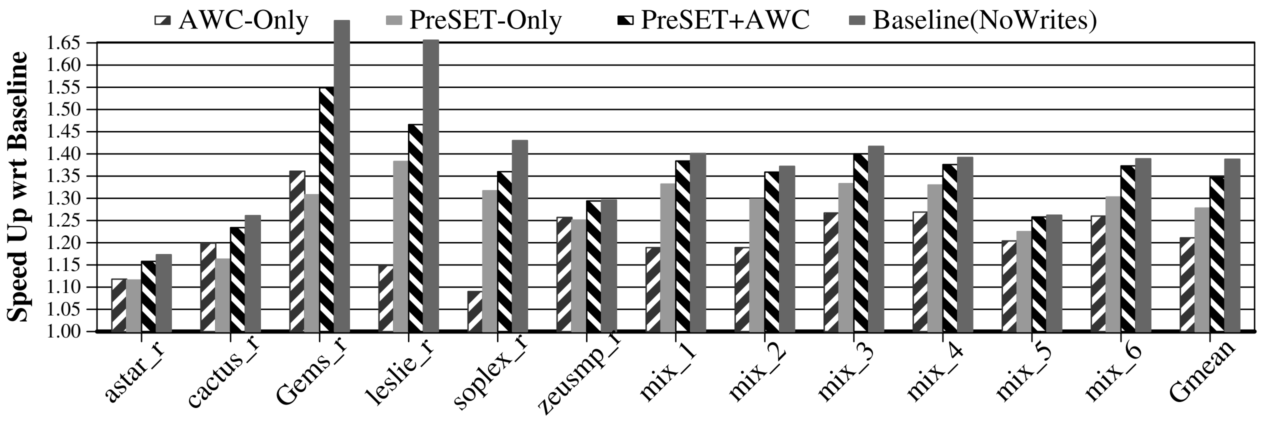

**Figure 9. Speedup with respect to Baseline for various systems**

## 5.3 Coverage of PreSET

If a PreSET operation is successful it allows the write operation to be done with lower latency. Ideally we want the PreSET operation for all writes to finish before the write reaches the memory system. However, PreSET is done only during idle periods and can get canceled many times in case there is heavy read/write traffic. Figure 10 shows the percentage of writebacks that are performed after PreSET is done.

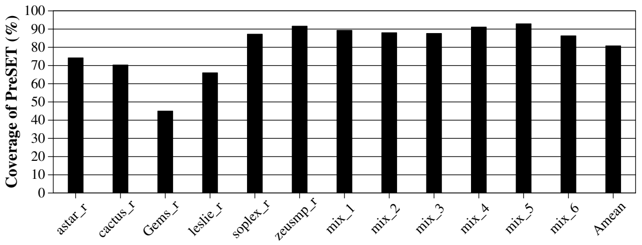

**Figure 10. Coverage of PreSET: Percentage of writebacks for which PreSET was already done**

On average 80% of the writes have PreSET done. For all benchmarks, except $Gems\_r$ majority of write access are done after PreSET. $Gems\_r$ has heavy write traffic and the idle period is less compared to other benchmarks. Therefore, even if the PreSET window is large (393 million cycles), there is frequent cancellation of PreSET requests and hence the coverage of less than 50%. For $mix$ workloads, a stream of heavy write request from one application can often find enough spare time in memory banks, if other workloads in the mix are not as memory intensive, hence the PreSET coverage is high.

## 5.4 Impact of System Write Bandwidth

Our proposal relies on having spare bandwidth in the memory banks to do PreSET. If the system has very little unused bandwidth then PreSET may not be as effective. We use a baseline that has 32 banks and utilization of approximately 30%. Figure 11 compares the performance of baseline, AWC, PreSET+AWC, and baseline with NoWrites as number of banks is varied from 16 to 64. All performance numbers are normalized with respect to baseline with 32 banks, and are averaged over all the workloads.

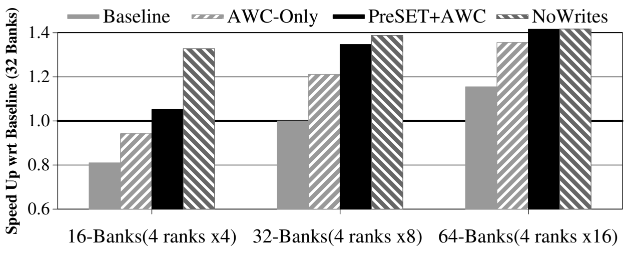

**Figure 11. Impact of changing the number of banks in memory on different PCM system**

When the number of banks is reduced to 16, the baseline performance degrades by 19%. Therefore, even though the average utilization of banks in our baseline (with 32 banks) is only 30%, having that many banks is useful. Even at 16 banks PreSET+AWC provides a speedup of 1.3x (1.05/0.81). However, there is lots of potential for improvement if all writes were removed (1.33/0.81=1.64x). Unfortunately, there is not enough bandwidth to do PreSET. When the number of banks is increased to 64, the contention from writes is reduced, hence the possible performance improvement reduces as well. Nonetheless, given the extra idle cycles in memory banks, PreSET+AWC gets all of the potential improvement. Increasing the number of banks in a PCM memory is not always practical though, because it is typically limited by maximum power consumption given that writes are power hungry operations. Therefore, doubling the number of banks would essentially double the number of concurrent write operations, doubling the power budget of PCM memory.

## 5.5 Impact on Write Queue Size and System Complexity

Our baseline assumes that each bank has a 32-entry WRQ. Each entry in the WRQ need to provision storage for both tag and data values. Therefore, having a 32-entry WRQ for each of the 32 banks, incurs a total storage overhead of 131KB. Furthermore, All WRQ entries associated with a bank are snooped on each read access, therefore designing a large WRQ incurs high complexity. Unfortunately, previously proposed AWC technique relies on having a large WRQ. Figure 12 compares the performance of baseline, AWC, Preset, PreSET+AWC, and baseline with NoWrites as the number of WRQ associated with each bank is varied from 8 to 256. All performance numbers are normalized to the baseline with a 32-entry WRQ and are averaged over all the workloads.

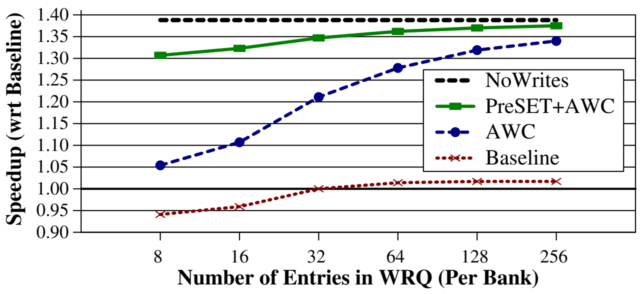

**Figure 12. Impact of WRQ size of effectiveness of AWC and PreSET+AWC. The performance of the baseline saturates at 64 entries. AWC needs large number of WRQ entries, whereas PreSET+AWC is relatively less sensitive to WRQ entries.**

For the baseline, reducing WRQ size from 32 to 8 reduces performance by 5%, whereas the performance saturates at a WRQ size of 64. The effectiveness of AWC is heavily dependent on the WRQ size. With a smaller 8-entry WRQ, the performance improvement of AWC reduces to only 5%. PreSET+AWC, on the other hand, is not so sensitive to the size of WRQ. This happens because PreSET converts most of the writes from long latency to short latency (a latency reduction of 8x), hence the write requests do not occupy WRQ for long time and therefore a smaller WRQ is sufficient. PreSET+AWC consistently obtains performance which is very close to a system with NoWrites.

Note that the WRQ overhead is incurred on a per-bank basis, therefore even for a 32-entry WRQ we would have a system with a total of 1024 WRQ entries where each entry must provision space for a cache line (128 bytes). Given the endurance and performance requirements the WRQ cannot be made in PCM, and would have to be made in significantly less dense technologies such as SRAM/eDRAM. Therefore, having storage for few thousand WRQ entries in the PCM controller becomes prohibitively expensive. In fact, given that PreSET+AWC is not so reliant on large WRQ, we can reduce the WRQ in our baseline system from 32 to 8, and still get 30% performance improvement compared to the baseline with 32-entry WRQ.

Unlike the WRQ, the PSQ is not snooped on every read and can be implemented as a set-associative structure instead of a CAM. Thus, PreSET+AWC not only improves performance but also reduce the area overhead and complexity in memory controller, obviating the need for large queue structures, and replacing them with smaller and simpler structures.

## 5.6 Impact of PSQ Size on PreSET

We use a default of 128-entry PSQ in our evaluations. As PSQ stores only address information and not data values, each PSQ entry is quite small (3 bytes). Therefore, the storage overhead of 128-entry PSQ is less than 400 bytes (10x lower compared to the 32-entry WRQ). We can provision an even larger PSQ so as to allow more PreSET requests to be buffered and hence increase the effectiveness of PreSET. Table 3 show the percentage performance improvement over baseline with PreSET+AWC as the number of entries in per-bank PSQ is varied from 8 to 512. The performance improvement is not very sensitive to PSQ size, especially beyond a 128-entry PSQ. Therefore, we use a 128-entry PSQ in our studies.

**Table 3. Effect of PSQ size on PreSET+AWC**

| Num PSQ entries | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|
| Speedup | 33.5% | 34.2% | 34.7% | 35.0% | 35.3% |

## 5.7 Impact of SET to RESET Ratio

In our studies we assume that SET incurs 8 times the latency of RESET. We analyze the sensitivity of our proposed scheme as this latency ratio is varied. For this analysis we assume that the SET latency is always 8 times longer than read latency, and vary the latency of only RESET operations. Table 4 shows the performance improvement of PreSET (alone) and PreSET+AWC, as the latency ratio of SET to RESET is varied from 2 to 8.

**Table 4. Effect of varying SET-to-RESET latency on performance improvement of PreSET and PreSET+AWC**

| SET-to-RESET Latency | 2X | 3X | 4X | 8X |
|---|---|---|---|---|
| PRESET-Alone | 17.3% | 22.7% | 24.9% | 27.8% |
| PRESET+AWC | 30.9% | 33.0% | 33.9% | 34.7% |

For comparisons, note that AWC improves performance by 21%, and the upper-bound (NoWrites) is 38%, and these improvements are not dependent on ratio of SET latency to RESET latency. For the standalone PreSET technique, the performance improvement is quite sensitive to SET-to-RESET latency, increasing from 17.3% at 2X latency ratio to 27.8% when latency ratio is 8X. PreSET+AWC, on the other hand, is not as sensitive to this ratio. For example,even at 4X latency ratio, the performance improvement of PreSET+AWC is still within 2% of that with 8X latency.

## 5.8 Bank Usage Distribution

The PreSET operation increases the number of cycles the bank spends in writing, with the aim of reducing the effective latency for reads. The extra writes caused by PreSET is essentially the overhead for our proposal. To analyze this, we divided the time a memory bank spends into four categories: reads, writes, PreSET, and idle. Figure 13 shows this distribution for baseline, AWC, PreSET, and PreSET+AWC, averaged over all the workloads. All numbers are normalized with respect to baseline.

The number of read cycles remain unchanged at 8%. The baseline spends 20% of the time in writes, and AWC increases it to 26%. With PreSET, the actual time doing the writes is reduced because writes become shorter latency (in common case). However, the bank utilization is increased because of PreSET operations. Note that because PreSET requests are non-blocking, they could be serviced multiple times for a given write, hence the large fraction of cycles spent in PreSET. When AWC is enabled with PreSET, it can cancel write operations, hence amount of time writing increases, offering less opportunities for PreSET. For PreSET+AWC, the total amount of time spent in writing (both demand as well as PreSET) increases from 20% to 40%. Given that the amount of time spent in writing has almost doubled, this can have a significant impact on power consumption and lifetime. The next section describes efficient variants of PreSET which reduce these overheads significantly while retaining most of the performance benefits.

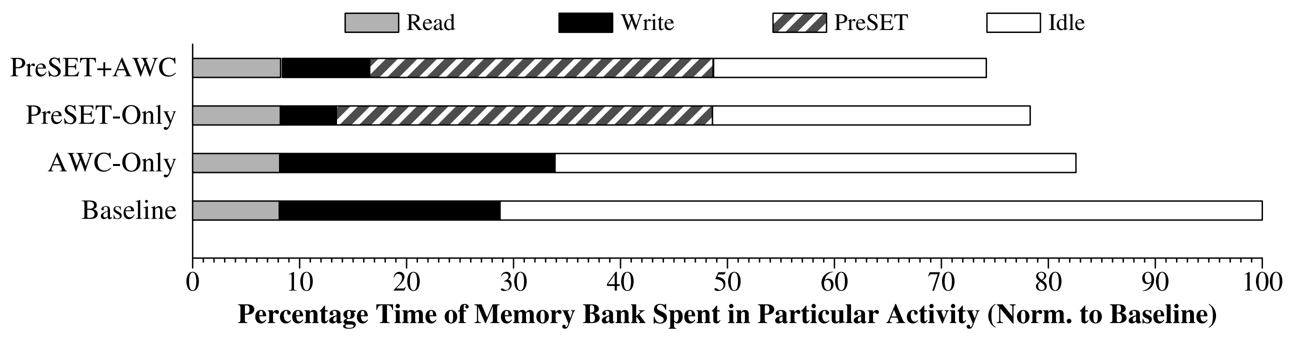

**Figure 13. Bank utilization for different PCM systems, normalized to baseline**

## 6 PreSET Throttling for Reduced Overheads

Ideally, we want to have PreSET operations only when it is likely to improve performance and avoid when alternative means of tolerating slow writes is sufficient. If the system already has AWC, then PreSET provides an advantage by converting long latency writes into short latency writes and thus avoiding the write queue to become full. However, AWC can tolerate occasional episodes of long latency writes, therefore it is not necessary to try to do PreSET for all dirty lines. Based on this insight we develop *PreSET Throttling*, which permits only a given percentage of all dirty lines to perform PreSET. PreSET throttling is regulated by the parameter *PreSET Drop Percentage (PDP)*, which denotes the percentage of PreSET requests squashed.

Recall that when a line is written to DRAM cache, the basic PreSET scheme tries to schedule a PreSET request for that line. If the PreSET request can be successfully inserted in the PreSET Queue, it sets the "PreSET Initiated" (PI) bit associated with the line to 1. To enable PreSET Throttling, when a PreSET request is initiated for a line, we consult a pseudo-random number generator that provides a value between 0 and 100. If the random value is less than PDP, then we do not send the PreSET request to the memory system and still set PI bit to 1. This also avoids sending subsequent PreSET requests for the line. Since the PreSET request was not sent, the "PreSET Done" (PD) bit associated with the line will remain at 0.[2] When such a line gets evicted, it will be serviced in a normal manner (without PreSET) and will rely on AWC for tolerating the long write latency. We describe two flavors of PreSET Throttling: Static and Dynamic.

### 6.1 Static PreSET Throttling

We can simply set the PDP parameter statically. We call this scheme *PreSET with Static Throttle (PreSET-ST)*. PreSET-ST allows us to smoothly choose between different data-points of performance benefits and overheads. Say, if we want to do PreSET only for about 60% of the lines, we would set PDP=40%. Given that we have AWC to handle occasional long latency writes, we would not expect the performance improvement of PreSET-ST+AWC to reduce linearly with PDP. And indeed this is what we observe in our evaluations. Figure 14 shows the performance improvement of PreSET-ST+AWC, and the percentage of time the memory bank is doing PreSET, as PDP is varied from 0% to 100%. At PDP=30%, the PreSET operations reduce by 30%, whereas the performance improvement changes negligibly (from 34.6% to 33.5%). When all PreSET requests are dropped (PDP=100%) the PreSET+AWC scheme degenerates into AWC, with performance improvement of 21% over the baseline. Given the negligible performance impact, we will use PDP=30% as the default value for all subsequent evaluations of PreSET-ST.

### 6.2 PreSET with Dynamic Throttling

We can also set the PDP parameter in PreSET Throttling dynamically based on the state of the system. We propose such a scheme called *PreSET with Dynamic Throttle (PreSET-DT)*, that

[2]We describe this implementation only for simplicity. If PreSET Throttling is implemented in this manner, then all lines where PreSET gets dropped will end up snooping the PSQ for invalidation. PSQ snooping can simply be avoided by using the impossible combination of bits (PI=0 PD=1) for denoting lines for which PreSET is dropped. With such an implementation, when a line with PI=0 and PD=1 is evicted, it will be serviced normally (without PreSET) without the need to snoop the PSQ.

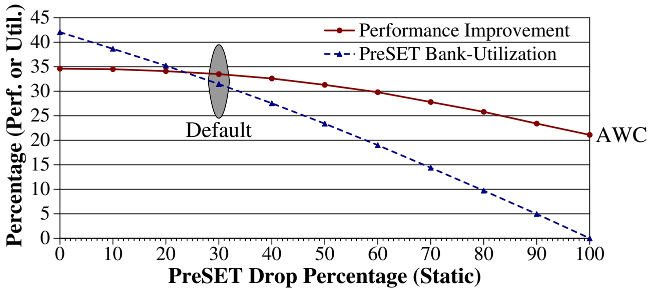

**Figure 14. Performance improvement and associated bank utilization of PreSET operations for PreSET-ST+AWC. The shaded oval represents the operating point of PreSET-ST+AWC with PDP=30% (default).**

determines PDP to be a function of the memory bank utilization. We measure the percentage of time the memory banks spends on write operations or PreSET operations in a given time quanta (five million cycles in our studies). We set PDP equal to this percentage at the end of the time quanta, as use this PDP for the next time quanta. The rationale behind selecting PDP as a function of memory utilization of write/PreSET is that if the memory system is doing a heavy amount of writes, then it would reach close to its power budget and would reach the endurance limited lifetime sooner than otherwise. Therefore, we do not want to exacerbate write traffic in such cases. However, if there is significant amount of spare bandwidth in memory system, then it would be better to do PreSET and obtain performance benefits. Our evaluations show that PreSET-DT+AWC provides an average performance improvement of 31% while having a bank utilization of 23%. While the PreSET-DT scheme is more conservative than PreSET-ST (30% PDP), the key advantage of PreSET-DI is that it limits power consumption and helps provide longer lifetime for write heavy workloads.

### 6.3 Impact on System Power and Energy

PreSET converts a given writes into two parts. First, an off-the-critical path PreSET, that performs only SET operations for all the bits in the line, and Second, a demand write which performs only RESET operations for some of the bits in the line. Such bifurcation causes extra bit flips, incurring extra power and energy. It is important to note though, that unlike RESET (which is a high power operation), PreSET only performs SET, which consume relatively much less power than RESET. Therefore, PreSET does not increase peak power consumption of memory system but it does increase average power consumption. Figure 15 shows the power, performance, energy, and energy-delay-product for different schemes. For evaluating PCM power, we use a model similar to Zhou et al. [17].

For power analysis, we break down the system power into three parts: PCM write power, PCM read power, and other components (DRAM cache and processor). For the PCM memory, most of the power is consumed in writing, and it accounts for 21% of overall system power. AWC increases overall system power to 1.33X and PreSET+AWC increases it to 1.38X. The throttling schemes are effective in limiting the power increase to 1.29X (static) and 1.27X (dynamic). Thus, throttling based schemes have approximately one-fourth lower power overhead than unconstrained version of PreSET+AWC, while retaining almost all of the performance ben-

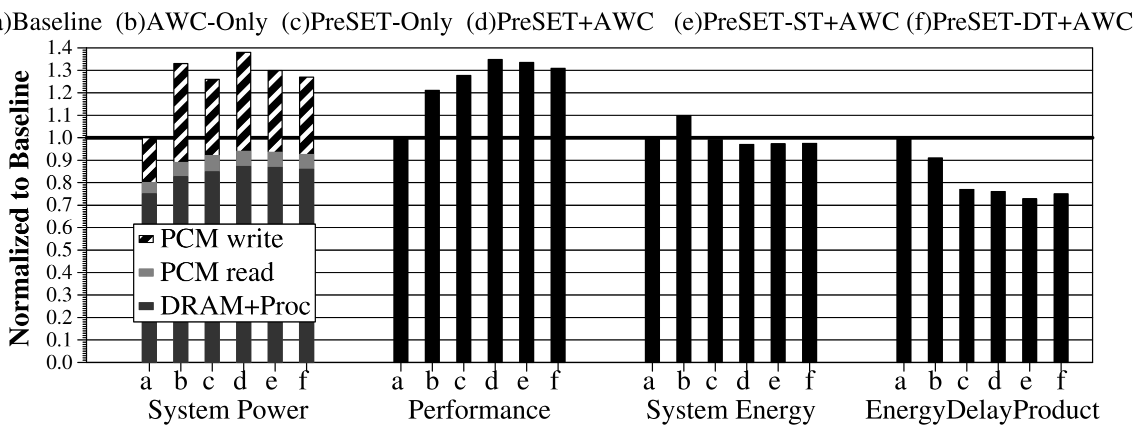

**Figure 15. Power-Performance comparison of different schemes. Note that PreSET-based schemes not only improve performance but also have significantly lower energy-delay product, much better than AWC-alone.**

efits. Given that all the PreSET-based proposals increase performance significantly the total energy remains similar to baseline, whereas the total energy with AWC is higher than baseline.[3]

Given that the system performs execution at a faster rate with PreSET-based schemes, the increase in power is not unexpected. However, we need a metric that can indicate the balance between performance and power. The Energy-Delay-Product (EDP) is one such metric which is often used as a combined figure-of-metric to compare different power-performance design points. Given that PreSET-based schemes reduce execution time significantly, they obtain much lower EDP. For the PreSET-only system EDP reduces by 22%, for the PreSET+AWC it reduces by 24%, with PreSET-ST+AWC it reduces by 27%, and with PreSET-DT+AWC it reduces by 25%. Comparatively, AWC reduces EDP by only 9%. Thus, PreSET-based schemes are attractive not only for improving performance but also for improving energy-delay product.

## 6.4 Impact on System Lifetime

The PreSET operation increases the number of write cycles which also impacts system lifetime. In this section, we compare the lifetime obtained by different schemes. We pessimistically assume that the PCM memory has a per-cell endurance of only 16 million writes (several studies [9][12][11] have assumed much higher endurance, which would make the lifetime problem even less severe). For our lifetime evaluations, we further assume perfect wear-leveling to simplify our analysis. Note that recent wear leveling algorithms such as Randomized Start-Gap [9] and Security Refresh [12] provide a lifetime very close (97%) to ideal wear leveling for typical workloads while incurring a write overhead of less than 1% [10] and storage overhead of less than 100 bytes, so our assumption of perfect wear leveling is not far from reality.

System lifetime is not only a function of endurance but also a function of the rate at which the PCM memory gets written. For example, if the baseline memory system is written continuously at 100% write traffic (without any read operations or idle time) then the system would last for approximately four years.[4] Figure 16 shows the lifetime of the baseline as a function of write bandwidth utilization. At 100% write traffic the system would last for 4 years and at 25% write traffic it would last for 16 years. Typical memory systems are designed to handle bursty accesses and to handle both read and write operations, so the episodes of continuous writes at full bandwidth is uncommon. For our workloads the memory write bandwidth utilization ranges from 16%-26%,[5] hence the lifetime of the baseline ranges from 16-24 years.

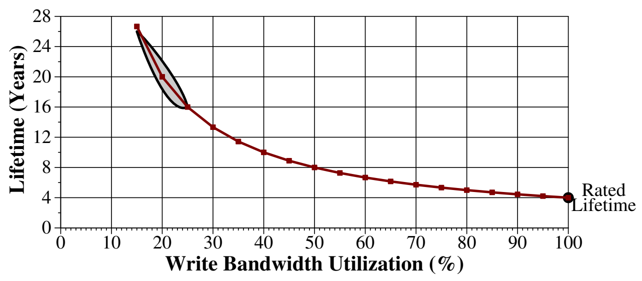

**Figure 16. Lifetime as a function of write bandwidth utilization. Our baseline system would be rated for four years assuming 100% writes. The write bandwidth utilization for our workloads is shown by the shaded oval.**

Figure 17 shows the system lifetime obtained for baseline, AWC, PreSET, PreSET+AWC, and PreSET with throttle. The bar labeled Amean denotes the arithmetic mean over all workloads. PreSET+AWC obtains a lifetime of 7.8 years on average, whereas with static throttle it becomes 8.6 years and with dynamic throttle it becomes 9.4 years. Thus, throttle based schemes help increase

[3]This happens because AWC incurs high power (for RESET operations) during the initial portions of write operations. When a write gets canceled this high power operation is repeated again, which causes significant power overheads. For example, even if a write gets canceled after 20% of its service is completed, the power consumption of this canceled write can be more than half of a completed write operation because of the initial high-power RESET operations. With PreSET, the final write that causes RESET gets completed with much shorter latency, thus avoiding the likelihood of re-executing the high power RESET operations.

[4]The number of lines per bank is $2^{23}$, each line can be written $2^{24}$ times, for a total of $2^{47}$ writes per bank. If $2^{20}$ writes per second can be performed for each bank, with approximately $2^{25}$ seconds per year, the system would last for $2^2$ years.

[5]One can increase write bandwidth utilization simply by reducing the number of banks. Unfortunately, it increases the memory bank contention as well. From Section 5.6, reducing number of banks of baseline from 32 to 16 causes approximately 20% slowdown.

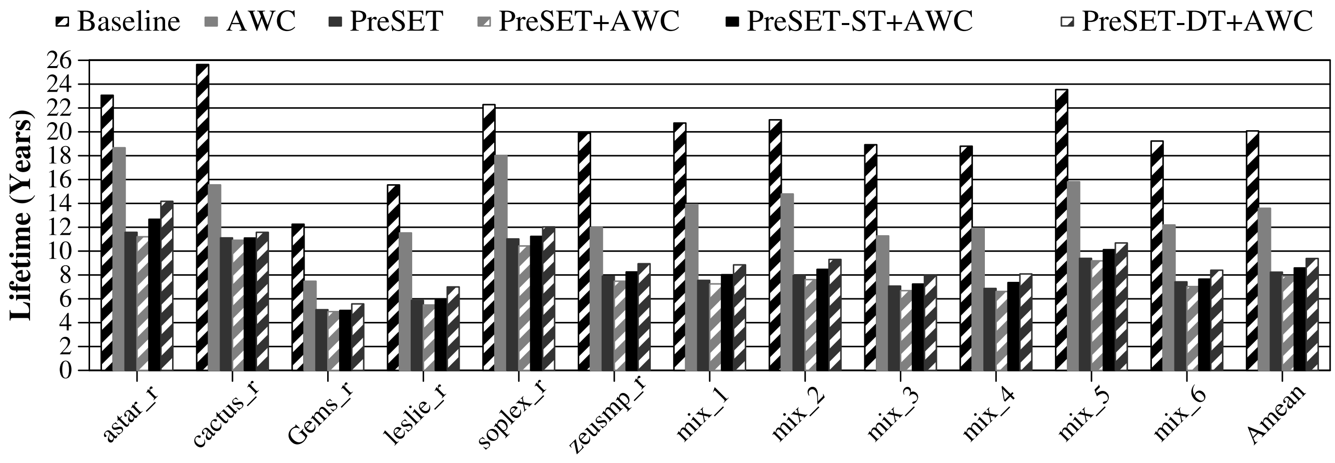

**Figure 17. Estimated system lifetime for different schemes, for a system with rated lifetime of four years.**

system lifetime. The key benefit of dynamic throttling scheme is to help improve lifetime of workloads that are write intensive, that would have much lower lifetime otherwise. For example, Gems_r obtains a lifetime of only 4.9 years with PreSET+AWC, whereas PreSET-DT+AWC increases it to 5.6 years.

As PreSET-based schemes cannot increase write-bandwidth utilization to more than 100%, the lifetime with PreSET-based techniques cannot be less than the rated lifetime of four years, regardless of the workload. Therefore, even for workloads that write continuously to memory, lifetime remains at four years, although as there would be no leftover bandwidth to do PreSET, there would be no performance/power benefits from PreSET.

## 7 Related Work

### 7.1 Related Concepts in Flash Domain

The notion that closely resembles the PreSET operation for PCM is the block Erase operation used in NOR and NAND Flash memories [2]. Similar to the SET and RESET operations in PCM, the Program (1→0) and Erase (0→1) operations in Flash memories have highly asymmetric latencies. Erase is typically one or more order of magnitude slower than Program. Given the high Erase latency, the Erase operation in Flash memories is performed over large blocks, typically 64-128 KB.

However, PreSET in PCM is fundamentally different from Erase in Flash memories for three reasons. First, PreSET is an optional operation, whereas Erase is a requirement before a particular block can be written in Flash memory. Second, the granularity of PreSET operation is the same as the unit of writing, whereas the unit of Erase is typically 16-64 times larger than the granularity of writes to Flash memories. Third, and fundamentally, PreSET is performed *in-place* and does not necessitate the use of large indirection tables which are needed in Flash memories because of *out-of-place* Program operations. Thus, PreSET avoids the significant area and latency overheads of large indirection tables.

For ensuring backwards compatibility with NOR Flash devices, some of the PCM products (e.g. Samsung part number K571229ACM) perform a block erase operation before the block can be written. The patent application by Lam et al. [3] describe a Flash-like block erasure scheme for PCM which initializes the block to a SET state, and then transitions only the required bits to the RESET state. However, no architecture technique is described on how to leverage this block erasure to enhance system performance. For example, applying erasure after the line reaches PCM memory ends up degrading system performance, and applying erasure before the line is written to results in data loss.

### 7.2 Related work in Writeback Scheduling

Other related work includes techniques for preventing bursty write traffic through intelligent write scheduling. *Eager Writeback* [5] speculatively cleans dirty lines in the last-level cache, by scheduling a writeback to memory for a given cache-line when the line is not expected to be written again prior to eviction. Unfortunately, an inaccurate prediction of last write results in extra write requests to memory. In comparison, PreSET request do not rely on speculation and can be sent at even the first write to a line. Furthermore, sending PreSET on first write instead of last write gets a much larger PreSET Window. Nonetheless, PreSET and Eager Writeback are complementary and can be combined.

The *Virtual Write Queues* [14] technique extends the idea of eager writeback, by performing clustered write operations to same DRAM page, in order to get page mode locality. While large granularity writes (several KB) are common in DRAM systems, they are unlikely to be present in PCM systems because of power efficiency reasons, thereby limiting the opportunity to do page mode writes in PCM memories.

The PreSET operation is complementary to the previous two techniques, and can be used in combination with them to improve system performance. The PreSET operation improves the physical write latency of a single write operation, and the prior techniques can be used to schedule the actual writeback operations, reducing bus and queue latencies.

### 7.3 Related Work in Reducing Bit-Flips

Zhou et al. [17] proposed redundant bit removal to reduce the unnecessary bit writes to PCM. It performs a read before write, and writes only the bits that have changed. As both types of transitions can still happen in the changing bits, the proposal does not reduce write latency. We can apply redundant bit removal for PreSET operations too, so that SET is performed only for the bits that are in RESET state. Furthermore, when a write happens after PreSET has completed, then only write the bits that cause RESET (without the need to do the read before the write).

Cho et al. [1] proposed *Flip-N-Write* to improve write power by reducing the number of bit transitions required while rewriting a previously written memory line. The key idea here is to store the data in either the original form or inverted form depending on which format reduces the number of bit transitions. We can apply Flip-N-Write to RESET operations for lines that are already PreSET, by writing the data in either normal or inverted form based on which causes fewer RESET transitions, thereby saving power.

## 8 Conclusion

The write operation in Phase Change Memory (PCM) is much slower than the read operation. Servicing such long latency write accesses can cause severe contention for read accesses, which results in significant increase in effective read latency and degradation in system performance. This paper exploits the fundamental property of PCM devices that writes are slow for only one type of data transition (SET) and are almost as fast as reads for the other transition (RESET), and makes the following contributions:

1. We propose *PreSET*, an architectural technique that proactively performs a SET operation for all the bits in a given memory line. Writing to a line that has been SET is much faster, and hence causes less contention for read requests.
2. We show that even a simple hardware based technique that initiates PreSET request on first write to cache line provides a high likelihood of PreSET getting completed before the corresponding write arrives to memory. We also discuss the hardware, memory controller, and interface support required to facilitate PreSET.
3. We evaluate PreSET and show that it provides better read latencies than adaptive write cancellation (AWC) policies without relying on large and complex hardware structures. PreSET when combined with AWC reduces the effective read latency of baseline from 982 cycles to 594 cycles. This provides an average performance improvement of 34%.
4. We also propose static and dynamic schemes for throttling PreSET operations. These schemes reduce the episodes of PreSET operations significantly while retaining a performance improvement similar to unconstrained version of PreSET. The proposal improves energy-delay-product by 25% and provides a system lifetime of 5+ years.

We believe that PreSET will be an important aspect of future PCM systems in order to tolerate the poor write performance of PCM memories. In this paper, we only analyzed systems that perform an in-place PreSET. A limited amount of out-of-place PreSET can help provide low write latencies, for scenarios where write latency is critical to performance (such as database transaction commit or persistent systems). Furthermore, we analyzed PreSET solely with the goal of improving system performance. PreSET policies that can adapt to energy consumption, power constraints, and lifetime can be designed as well. Exploring such extensions is a part of our current and future work.

## Acknowledgments

This paper is dedicated to the memory of our colleague and mentor John P. Karidis. John was an exceptionally versatile researcher, an unstoppable idea fountain, and a great human being. Among other things, John was the technical lead and the initiator of the PCM-based Hybrid Memory project at IBM. All the co authors greatly thank him for his years of mentorship and guidance. We will miss you John.

## References

[1] S. Cho and H. Lee. Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance. MICRO 42, pages 347–357, 2009.

[2] L. Crippa, R. Micheloni, I. Motta, and M. Sangalli. Nonvolatile memories: Nor vs. nand architectures. In *Memories in Wireless Systems*, pages 29–53. Springer Berlin Heidelberg, 2008.

[3] C. Lam et al. Block erase for phase change memory. United States Patent Application 20090027950.

[4] B. Lee et al. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA-36*, 2009.

[5] H. Lee et al. Eager writeback - a technique for improving bandwidth utilization. In *MICRO-2000*.

[6] K.-J. Lee et al. A 90nm 1.8v 512mb diode-switch pram with 266mb/s read throughput. In *IEEE Journal of Solid-state Circuits*, 2008.

[7] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA-36*, 2009.

[8] M. K. Qureshi et al. Improving read performance of phase change memories via write cancellation and write pausing. In *HPCA-16*, 2010.

[9] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO-42*, pages 14–23, 2009.

[10] M. K. Qureshi, A. Seznec, L. Lastras, and M. Franceschini. Practical and secure pcm systems by online detection of malicious write streams. In *HPCA*, pages 478–489, 2011.

[11] S. Schechter, G. H. Loh, K. Strauss, and D. Burger. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA-37*, 2010.

[12] N. H. Seong, D. H. Woo, and H.-H. S. Lee. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA-37*, 2010.

[13] A. Snavely, D. M. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *SIGMETRICS*, pages 66–76, 2002.

[14] J. Stuecheli et al. The virtual write queue: coordinating dram and last-level cache policies. ISCA-2010.

[15] J. Tominaga et al. Structure of the Optical Phase Change Memory Alloy, AgVInSbTe, Determined by Optical Spectroscopy and Electron Diffraction,. *J. Appl. Phys.*, 82(7), 1997.

[16] N. Yamada, E. Ohno, K. Nishiuchi, and N. Akahira. Rapid-Phase Transitions of GeTe-Sb2Te3 Pseudobinary Amorphous Thin Films for an Optical Disk Memory. *J. Appl. Phys.*, 69(5), 1991.

[17] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA-36*, 2009.