Balancing Performance and Lifetime of MLC PCM by Using a Region Retention Monitor

Mingzhe Zhang*†, Lunkai Zhang§, Lei Jiang¶, Zhiyong Liu*‡ and Frederic T. Chong§
*Beijing Key Laboratory of Mobile Computing and Pervasive Device, ICT, CAS, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡State Key Laboratory of Computer Architecture, ICT, CAS, Beijing, China
§Computer Science Department, University of Chicago
¶Department fo Intelligent Systems Engineering, Indiana University Bloomington
{zhangmingzhe, zyliu}@ict.ac.cn, lunkai@uchicago.edu, jiang60@iu.edu, chong@cs.uchicago.edu

***Abstract*—Multi Level Cell (MLC) Phase Change Memory (PCM) is an enhancement of PCM technology, which provides higher capacity by allowing multiple digital bits to be stored in a single PCM cell. However, the retention time of MLC PCM is limited by the resistance drift problem and refresh operations are required. Previous work shows that there exists a trade-off between write latency and retention—a write scheme with more SET iterations and smaller current provides a longer retention time but at the cost of a longer write latency. Otherwise, a write scheme with fewer SET iterations achieves high performance for writes but requires a greater number of refresh operations due to its significantly reduced retention time, and this hurts the lifetime of MLC PCM.**

In this paper, we show that only a small part of memory (i.e., hot memory regions) will be frequently accessed in a given period of time. Based on such an observation, we propose *Region Retention Monitor (RRM)*, a novel structure that records and predicts the write frequency of memory regions. For every incoming memory write operation, *RRM* select a proper write latency for it. Our evaluations show that *RRM* helps the system improves the balance between system performance and memory lifetime. On the performance side, the system with *RRM* bridges 77.2% of the performance gap between systems with long writes and systems with short writes. On the lifetime side, a system with *RRM* achieves a lifetime of 6.4 years, while systems using only long writes and short writes achieve lifetimes of 10.6 and 0.3 years, respectively. Also, we can easily control the aggressiveness of *RRM* through an attribute called *hot_threshold*. A more aggressively configured *RRM* can achieve the performance which is only 3.5% inferior than the system using static short writes, while still achieve a lifetime of 5.78 years.

***Keywords*-Non-Volatile Memory; Phase Change Memory; Dynamic Trade-off;**

I. Introduction

Phase Change Memory (PCM) technology is considered a promising replacement for DRAM technology in main memory systems due to its advantages of high scalability, high density, low dissipation power and non-volatility. In addition, PCM belongs to the resistive memory family, which uses different resistance values to record data. This characteristic gives PCM the ability to further increase storage capacity. For example, using a fine-grained definition of the intermediate resistance value, one PCM cell manages to store multiple digital bits. PCM using such a technology is called Multi Level Cell (MLC) PCM.

Although MLC PCM has a capacity advantage compared to Single Level Cell (SLC) PCM, it also has a cost: MLC PCM has a shorter retention time (up to a few hours) because of the resistance drift problem. As a result, MLC PCM needs to be refreshed at a lower frequency compared with DRAM technology. Moreover, write precision also influences the retention time. Higher precision writes, which needs more SET iterations and thus a longer write latency, can lead to a longer retention time. As a result, there exists a write latency/retention trade-off in MLC PCM—longer writes have longer retention times. With the model introduced in Section II, a write with 7 SET iterations has a retention time of 3054.9 seconds, while a write with 3 SET iterations has a retention time of only 2.01 seconds.

Basically, we have two static write modes: universally using long-latency-long-retention writes or just using short-latency-short-retention writes. However, neither option is ideal. The former option has considerable performance degradation caused by a longer write latency; the latter option has an unacceptable PCM lifetime because of the frequently needed refreshing operations.

In this paper, we propose a hybrid write scheme to avoid the drawbacks while preserving the advantages of both static write modes. We show that, due to the locality of applications, usually only a small part of memory (i.e., hot memory regions) will be frequently accessed in a given period of time. Therefore, we use *Region Retention Monitor (RRM)*, a novel structure between the Last Level Cache (LLC) and memory controller to dynamically identify current hot memory regions and then only perform short-latency-short-retention write operations to the addresses associated with these regions. Our experiment shows that, with minimal hardware overhead and complexity, *RRM* helps the system achieve a better balance between performance and lifetime.

Lunkai Zhang and Mingzhe Zhang have equal contribution. This work was performed while Mingzhe Zhang was a visitng Ph.D student at Computer Science Department, the University of Chicago.

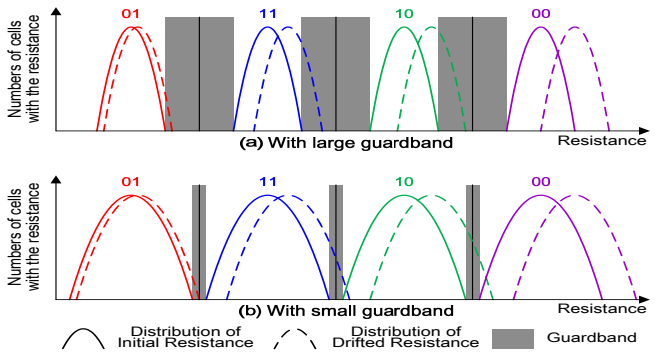

Figure 1. The problem of resistance drift in MLC PCM.

The rest of the paper is organized as follows. In Section II, we introduce the background of the write latency/retention trade-off in MLC PCM. In Section III, we show the drawbacks of static write modes, and the possibility of using a hybrid write scheme to overcome the problems. After that, Section IV introduces our proposed *RRM* architecture. Then we present the experiment methodology and results in Sections V and VI. Finally, a detailed related work section is presented in Section VII, followed by the conclusions in Section VIII.

II. MLC PCM and its Write Latency vs. Retention Trade-off

Though PCM is said to be "non-volatile," the resistance of a PCM cell actually does not remain the same forever. Due to the chalcogenide structural relaxation [1], the resistance of PCM spontaneously increases, which is called resistance drift. Such resistance drift is not a big problem for Single Level Cell (SLC) PCM, which just stores one digital bit in a cell, because of a remarkable resistance difference in a range between values "1" and "0". It is, however, a serious concern for MLC PCM due to the reduced resistance range, which presents as a digital value. Once the current resistance leaves the resistance range of its represented digital value, the data will be lost. A natural way to solve this problem is to leave an unused resistance range between two adjacent digital values, which is called a *guardband*. As shown in Figure 1, a *guardband* works by ensuring that the drifted resistance after a given period still sits within the resistance range of its represented digital value. The larger the *guardband* is, the longer retention (i.e., the period during which the stored data are correct) a MLC PCM cell can achieve.

Though a larger *guardband* can guarantee longer data retention, it is achieved at the cost of a longer write latency. Larger *guardband* means the initial resistance should be within a narrower (i.e., more precise) resistance range. An MLC PCM write is usually composed of one RESET and multiple SET iterations, and the narrower the resistance range, the more SET iterations are needed in order to achieve the desired resistance precision; thus a longer total MLC PCM write latency is required. As a result, the overall write latency is increased as we increase the *guardband* size.

Table I
THE WRITE LATENCY AND RETENTION OF WRITE OPERATIONS WITH DIFFERENT NUMBER OF SET ITERATIONS.

Write Type	Current (in μA)	N. Energy	Retention (in s.)	Latency (in ns.)
7-SETs-Write	30	1	3054.9	1150
6-SETs-Write	32	0.975	991.4	1000
5-SETs-Write	35	0.972	104.4	850
4-SETs-Write	37	0.869	24.05	700
3-SETs-Write	42	0.84	2.01	550

Here we model in detail the trade-off between write latency and data retention in MLC PCM. We adopt the write latency/retention model of Li et al.'s work [2], which is based on models from various aspects, e.g., the PCM current/resistance model [3], the process variation model [4], the resistance drifting model [1] and the write energy/cell endurance model [5]. We re-calculate these with PCM cell parameters and read/write parameters from the latest 20 nm PCM chip demonstration [6], and show the detailed modeled results in Table I. A MLC PCM write begins with one RESET and then continues with several SET iterations. A RESET operation takes 100ns and 50μA current irrespective of the number of SET iterations that follow. A SET operation takes 150ns, and its current varies with the number of needed SET iterations—with fewer SET iterations in an MLC PCM write, we have to use a higher SET current to reach the target resistance range faster. Writes with more SET iterations can achieve better retention—a 7-SETs-Write manages to achieve a retention time of nearly one hour (3054.9 seconds), whereas a 3-SETs-Write has a retention time of only 2.01 seconds. Although a longer retention is achieved by using more SET iterations, it comes with a cost of a longer total MLC PCM write latency: a 7-SETs-Write requires more than twice the latency of a 3-SETs-Write.

As discussed in Kim and Ahn's work [5], the SET operations have little impact on PCM cell endurance, while RESET operations are the dominant factor that determines the endurance of a PCM cell. Therefore, all MLC PCM cells here can achieve the same endurance no matter which write mode is used.

III. Motivation

In this section, we first show that having a single write latency in MLC PCM main memory will cause performance degradation or an unacceptable short lifetime. Then we discuss why state-of-the-art solutions are ineffective in resolving problem. Finally, based on experimental data, we present the opportunity for a pure hardware solution for this problem.

A. Imperfection of Using a Fixed Number of SETs in All Write Operations

As we discussed in Section II, the number of SETs in a write operation decides its write latency and data retention. And the write latency and data retention of write operations in turn affect the performance and PCM lifetime of the overall system:

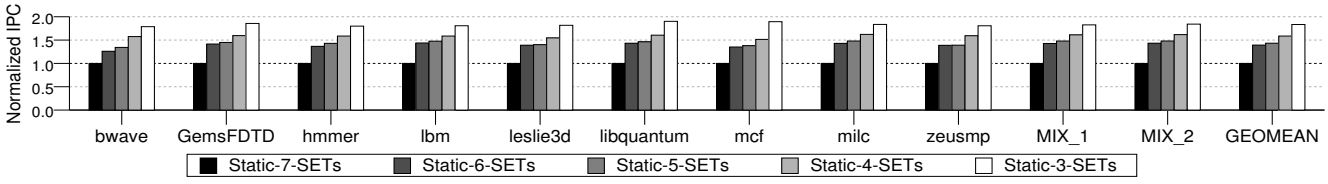

Figure 2. **Normalized IPC of static write schemes.**

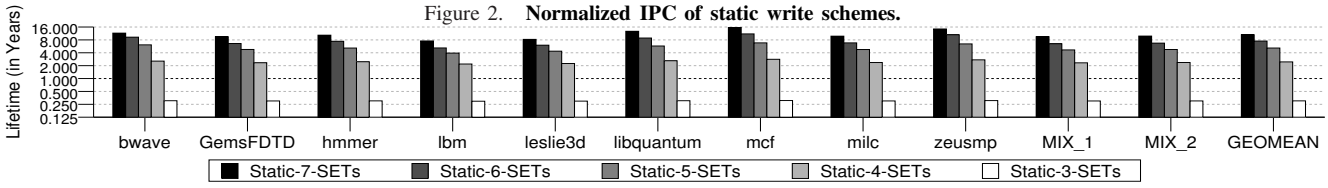

Figure 3. **Memory lifetime of static write schemes.**

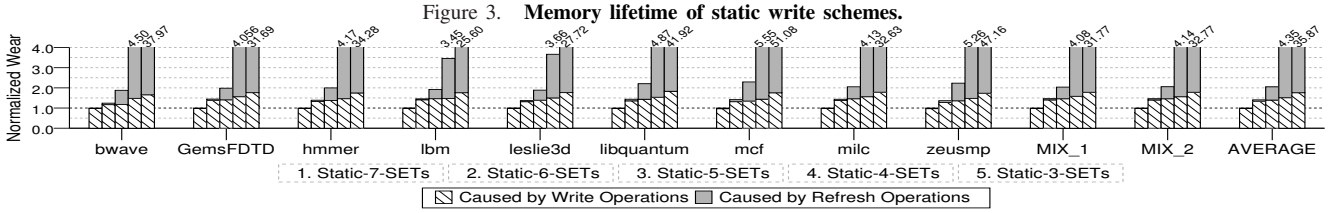

Figure 4. **Normalized wear introduced by write and refresh operations of static write schemes.**

Table II
QUALITATIVELY COMPARISON OF THE SCHEMES USING THE WRITE LATENCY VS. RETENTION TRADE-OFF IN MLC PCM.

Schemes	Target System	Support Dynamic Program	Require Secondary Storage	Support Processor Caches	Compatible With General-Purpose OS	Hardware Overhead	Fine Granularity Management
CDDW[7]	Embeded System Memory	No	No	No	No	Low	Yes
WMALT[8]	Embeded System Memory	No	No	No	No	Low	Yes
MMAS[9]	Embeded System Memory	No	No	No	No	Mid-High	Yes
Amnesic Cache [10]	File Cache	Yes	Yes	N/A	N/A	High	Yes
NVM Duet [11]	General-Purpose Memory	Yes	No	Yes	No	Low	No
Region Retention Monitor	General-Purpose Memory	Yes	No	Yes	Yes	Low	Yes

- **Lifetime.** Though MLC PCM writes with different SETs have almost the same endurance (as discussed in Section II), they require a different refresh frequency. As a result, write operations with fewer SETs need more frequent refresh (i.e., write) operations, which result in a short PCM lifetime.
- **Performance.** The number of SETs in MLC PCM writes has two opposite effects on the system performance: first, fewer SETs lead to a shorter write latency and thus improve the performance; second, fewer SETs lead to more frequent refreshing operations, which cause memory halt and thus may degrade performance.

A natural question is whether we can use writes with a fixed number of SETs to achieve good performance and acceptable lifetime. To answer this question, we simulate systems (see Section V for detailed configurations) with a different fixed number of SETs in write operations in their MLC PCM main memory system. Note that to simplify our experiment setup, we do not take into account the performance degeneration caused by refresh operations—as a result, a system using write operations with few SETs (e.g., 3-SETs-Writes and 4-SETs-Writes) actually performs considerably worse than what we report here. The results are shown in Figures 2 to 4. From this, we can make the following conclusions:

- Not surprisingly, fewer SETs lead to higher performance. In particular, a system with 3-SETs-Writes has unmatched performance—as a geomean, it outperforms the second best system (the one with 4-SETs-Writes) by 15.6%, and the performance benefit is up to 90.1% in `libquantum`.
- However, fewer SETs lead to more frequent refreshing. For a system with 3-SETs-Writes, this overhead is unacceptable—even just considering the wear of refreshing operations, such a system has a lifetime of only 0.317 years.

Ideally, we want a system that has performance similar to a system using 3-SETs-Writes, but also achieves lifetime similar to a system using 7-SETs-Writes.

B. Ineffectiveness of State-of-the-Art Solutions

Some previous proposals also work on improving the performance and lifetime of MLC PCM by using the same trade-off between write latency and data retention. However, these approaches all work in different scenarios and cannot be adapted to our target case, which is the MLC PCM main memory of a general-purpose computing system. As example, here we discuss five closely related proposals: CDDW[7], WMALT[8], MMAS[9], Amnesic Cache [10] and NVM Duet [11].

The first three solutions (CDDW[7], WMALT[8] and MMAS[9]) use *Directed Acyclic Graph* (DAG) in the applications to statistically estimate their data lifetime and then use different write operations accordingly. Although clever, the use of these schemes is very limited: first, they only work for the static applications that can be represented by *DAG* and cannot be used for dynamic programs; second, they hardly work for systems with caches because the filtering effect of caches makes the actual PCM memory operations distinct from application read/write operations; third, the memory management of operating systems (OSes) may make memory operations even more different from the prediction of DAG. Since caches and OSes are both integral parts of general-purpose systems and dynamic programs are also important applications for general-purpose systems, these DAG-based approaches hardly work for general-purpose systems.

Recently, Kang et al. [10] proposed Amnesic Cache as an MLC PCM-based file cache in front of an SSD/hard disk. The goal of Amnesic Cache is to reduce the access latency of its MLC PCM. Its approach is to write all the blocks with writes that have few SETs (thus a short retention and a short write latency) first and then promote some frequently accessed blocks with writes that have more SETs (thus a longer retention and a longer write latency). Amnesic Cache is not suitable for MLC PCM-based main memory for three reasons: first, Amnesic Cache must work with secondary storage because it needs to write back the blocks whose retention time is expired; second, its promotion write schemes hurt the MLC PCM lifetime by issuing multiple writes to the same block; third, it requires nontrivial memory storage to record the access history of all cache blocks. Amnesic Cache works fine as file caches that have SSD secondary storage does not require high endurance due to infrequent write operations and have abundant memory storage. However, these three attributes are not compatible when our target is MLC PCM based main memory. There is also a proposal [12] to speed up the write operations of MLC NAND Flash through short-latency-short-retention writes. Similar to Amnesic Cache, this proposal requires quite complex retention tracking mechanisms that cannot be adopted in the main memory controller for hardware overhead and timing reasons.

Liu *et al.* presented a solution called NVM Duet that provides a unified storage architecture for both memory and persistent storage[11]. In NVM Duet, PCM memory cells are adaptively used as memory or persistent storage and written with different write modes accordingly (short-latency-short-retention writes for memory, and long-latency-long-retention writes for persistent storage). However, NVM Duet can only choose a single write mode for one application. As we know, the memory access behaviors are not universal inside an application (as shown in Table III), and NVM Duet cannot use such behavior differences inside an application. Also, NVM Duet requires nontrivial OS modification and thus it is incompatible with the state-of-the-art commercial OSes.

Table III
DETAILED TEMPORAL AND SPACIAL WRITE BEHAVIORS OF GEMSFDTD, WE SIMULATE 4 COPIES OF GEMSFDTD ON 8GB MLC-PCM MAIN MEMORY FOR 5 SECONDS, AND RECORD THE WRITE BEHAVIOR IN 4KB-GRANULARITY REGIONS.

Benchmark GemsFDTD				
Average Write Interval	# Regions	% of Regions	# Writes	% of Total Writes
$< 10^6 ns$	44	0.0%	1277 K	0.9%
10^6ns to 10^7ns	22865	1.1%	109862 K	76.64%
10^7ns to 10^8ns	5569	0.3%	23314 K	15.6%
10^8ns to 1s	979	0.0%	3024 K	2.1%
1s to 2s	12021	0.6%	3044.9 K	2.1%
written once	3826	0.2%	3824 K	2.7%
never written	2051848	97.8%		

Table II shows a qualitative comparison of state-of-the-art schemes using the write latency vs. retention trade-off in MLC PCM. The attributes in blue are the desirable ones for MLC PCM main memory in a general-purpose system, while the red ones are undesirable. We can see that none of the previous proposed schemes fulfills the requirements. Therefore, to achieve a long lifetime and high performance for MLC PCM main memory in a general-purpose system, we need to come up with some new solutions.

C. Our Observation

Here we investigate the write behaviors in granularity of 4KB memory regions of a baseline system using just *7-SETs-Writes*, which are slow but have quite a long retention. We run the simulation with the configuration described in Section V represent the statistics for `GemsFDTD` in Table III. We can conclude that a fairly limited fraction of memory (about 2%) gets most of the write operations (up to 97.3%). We call these memory regions *Hot-Written Memory Regions* and the rest of the memory regions *Cold-Written Memory Regions* . Therefore, we can conduct fast and short-retention writes (e.g., *3-SETs-Writes*) to *Hot-Written Memory Regions* and use default slow and long-retention writes (*7-SETs-Writes*) to *Cold-Written Memory Regions* . Therefore, the memory system performance will be considerably improved because most of the writes are transferred to fast writes. At the same time, because *Hot-Written Memory Regions* are relatively few, the need for frequent memory refreshing will be remarkably reduced compared to a scheme with all fast writes. This, in turn, improves the memory lifetime. In the next section, we will introduce our detailed scheme: *Region Retention Monitor*.

IV. REGION RETENTION MONITOR

Based on the observation made in Section III-C, we propose *Region Retention Monitor (RRM)* which identifies the current frequently written memory regions and conducts fast but short retention writes to them. For the rest of memory, RRM conducts slow but long retention writes. RRM also issues selective refresh requests to ensure the correctness

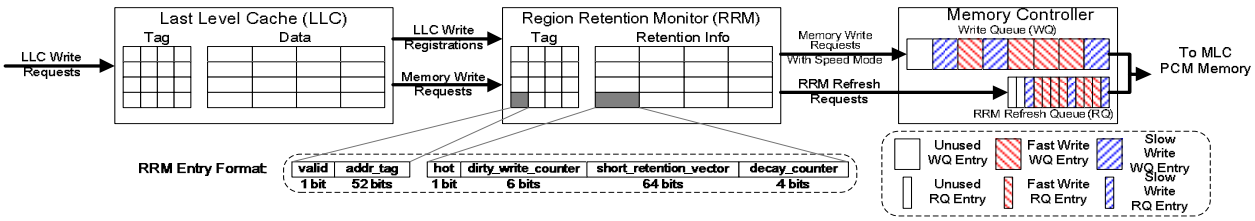

Figure 5. A high-level view of *RRM* and its relationship to the Last Level Cache (LLC) and memory controller.

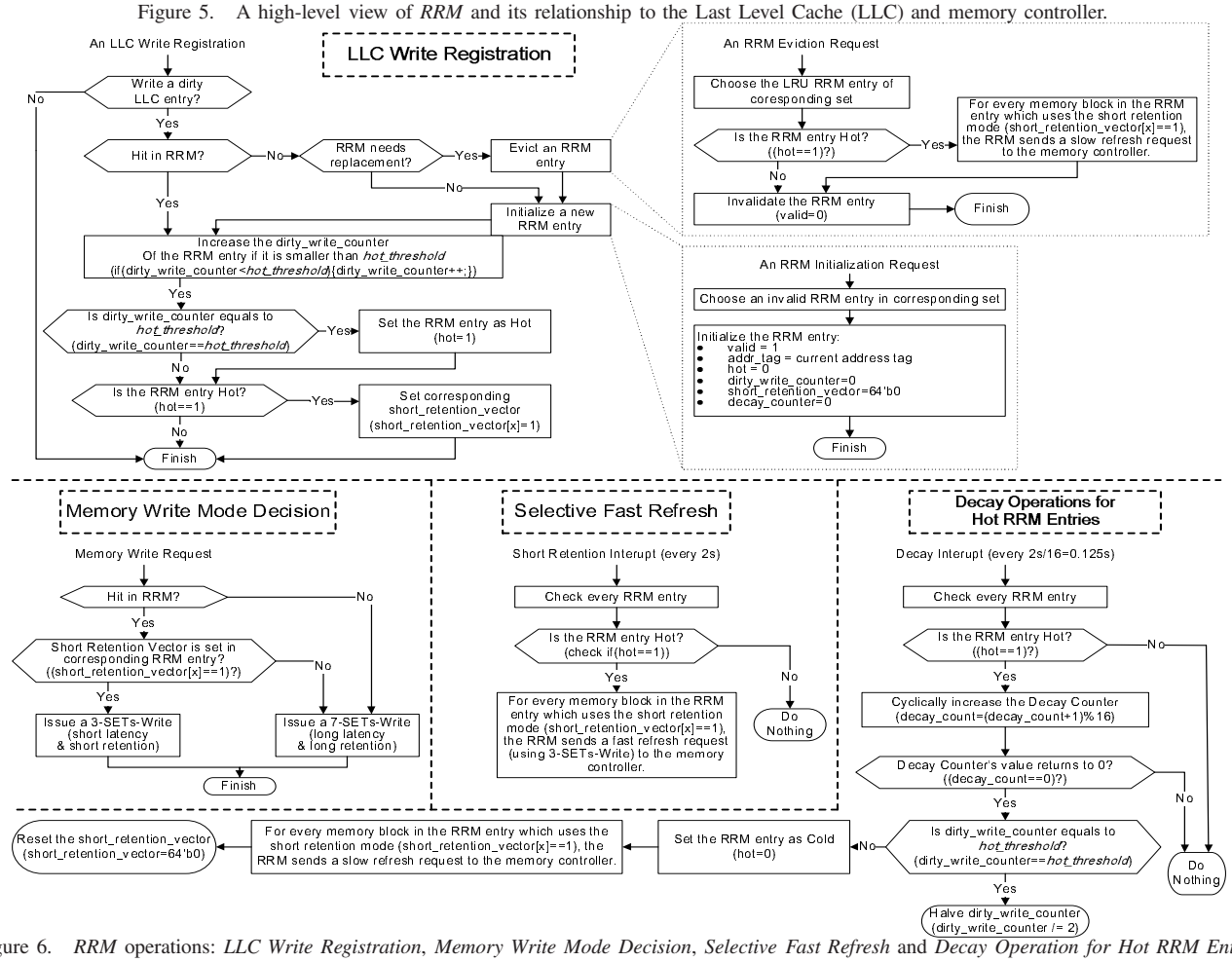

Figure 6. *RRM* operations: *LLC Write Registration*, *Memory Write Mode Decision*, *Selective Fast Refresh* and *Decay Operation for Hot RRM Entries*.

of data. As a result of its adaptive write mode selection to difference writes, RRM balances the performance and lifetime of MLC PCM memory. In the rest of this section, we will introduce RRM in detail.

A. RRM Overview

As shown in Figure 5, RRM lies between the LLC and memory controller. It monitors the operations of LLC and decides which write mode the current memory request should use. (For implementation simplicity, we only consider two write modes: *3-SETs-Write* or *7-SETs-Write*.) RRM is also in charge of issuing specific refresh requests other than normal long retention requests.

RRM requires additional modifications to the LLC and memory controller. On one hand, upon every LLC write operation, the LLC sends to RRM an LLC Write Registration message, which includes the address information and whether the written LLC entry is previously dirty (to filter out streaming writes, see Section IV-D). On the other hand, the memory controller needs to adopt two different write modes (*3-SETs-Write* or *7-SETs-Write*), and it also needs a new RRM Refresh Queue, which holds the refresh request sent by RRM.

Just like a low-level cache, RRM is managed in a set-associated manner, and its storage is divided into two separated arrays: Tag Array, which stores the address information and valid identifier, and Retention Information Array, which stores the detailed information to guide the write mode selection.

B. Inputs and Outputs

RRM has two types of input (i.e., *LLC Write Registrations* and *Memory Write Requests*) and two types of output (i.e.,

Memory Write Requests with Speed Mode and *RRM Refresh Requests*), as shown in Figure 5. An *LLC Write Registration* is similar to an *LLC Write Request* except that it has additional information to indicate whether the LLC write is to a dirty LLC entry or not. For every *Memory Write Request* input from LLC, the RRM decides which write mode it should use (*3-SETs-Write* or *7-SETs-Write*) and then forwards the requests to the memory controller as *Memory Write Requests with Write Mode*. RRM also needs to issue selective refresh requests (i.e., *RRM Refresh Requests*) to ensure the correctness of memory data.

C. RRM Entry Format

Each RRM entry records the write behavior of an aligned address region (i.e., Retention Region), and it also stores the write mode decision of the corresponding Retention Region. In our paper, the size of each Retention Region is 4KB—the same size of a general OS page. As shown in Figure 5, an RRM entry contains the following fields:

- *Valid (1 bit),* which indicates the validation of the entry
- *Addr (52 bits),* which records the address information (64 full address bits but ruling out 12 in-region address bits)
- *Hot (1 bit).* If set to 1, it indicates the corresponding Retention Region is *hot* (frequently written); otherwise, the corresponding Retention Region is *cold* (infrequently written).
- *Dirty_write_counter (6 bits).* This counter records the number of writes to dirty LLC entries in the region. Note that only the writes to dirty LLC entries (instead of writes to all LLC entries) are recorded in order to rule out the possibility of streaming writes pollution (see Section IV-D). If the counter equals *hot_threshold* (by default, *hot_thresh-old*==16), the corresponding Retention Region will be set as hot (*hot=1*).
- *Short_retention_vector (64 bits),* which stores the write mode decision for every memory block (64 bytes per memory block and 64 memory blocks per Retention Region). If the corresponding *short_rete-ntion_vector* bit is 1, then the memory block will be written with *3-SETs-Write* (fast but short retention writes); otherwise, it will be written with default *7-SETs-Write* (slow but long retention writes).
- *Decay_counter (4 bits),* a cyclic counter that is used to decay obsolete *hot* Retention Regions to *cold*.

D. LLC Write Registration

LLC Write Registration refers to the procedure in which RRM gathers behavior information from LLC write operations. A write registration message from LLC includes the write address and whether it writes the dirty (previously written) LLC entry. Based on such information, RRM conducts the following operations (see the top portion of Figure 6):

- *The RRM continues the write registration only when the LLC write is to a dirty LLC entry.* In the end of this subsection, we discuss in detail why we only consider LLC writes to dirty LLC entries, and ignore the ones to clean LLC entries.
- *RRM entry lookup, eviction, and initialization.* These are the typical entry lookup operations of a cache-like structure. We first find the hit RRM entry according to the address, and then initialize a new RRM entry if there is no hit RRM entry. If there is no free RRM entry for initialization, we need to evict the LRU RRM entry in the corresponding set to make room for the new entry.
- *Increasing dirty_write_counter if it is smaller than hot_threshold. Then if it equals to hot_threshold, RRM sets the entry as hot.*
- *If the entry is hot, then set corresponding bit of current write address in short_retention_vector to 1.* This indicates that the future memory write for the current address will use *3-SETs-Write* (fast and short retention write).

Note that, we only consider the LLC writes to dirty LLC entries in order to filter out streaming write patterns. If we also register the writes to clean LLC entries, a write region with a streaming write feature (i.e., *very low temporal write locality* but *strong spatial write locality*) can also easily accumulate enough writes to set the corresponding Retention Region as *hot*. Recall from Section III-C that the purpose of the RRM scheme is to identify the memory blocks with high temporal write locality. Thus including the clean writes is against the purpose of RRM and should be avoided.

E. Memory Mode Decision

The bottom-left portion of Figure 6 shows the Memory Mode Decision operations of RRM. On a memory write request, RRM first checks whether there is any hit RRM entry. If there a hit RRM entry with the corresponding bit in *short_retention_vector* set to 1, then we issue a *3-SETs-Write* (fast and short retention write) to MLC PCM memory; otherwise we issue a *7-SETs-Write* (slow and long retention write).

F. Selective Fast Refresh

The bottom-middle portion of Figure 6 shows the Selective Fast Refresh operations of RRM. On a short retention interrupt (with a 2-seconds interval, which is 0.01 seconds shorter than the retention time of *3-SETs-Write*), for every RRM entry, RRM first checks whether it is *hot*. If it is, for every memory block in the RRM entry that uses the short retention mode (*short_retention_vector[x]==1*), the RRM sends a fast refresh request (using *3-SETs-Write*) to the memory controller. Compared with memory write requests, we give higher priority to these selective fast refresh requests because they have a relatively tight time requirement (0.01 seconds before the data retention time is over).

Note that RRM does not issue global refresh requests, which use *7-SETs-Writes* and need to be issued every 3054

seconds or so for all memory blocks. This is because we assume that the MLC PCM has a built-in self-refresh circuit that automatically handles the global refresh requirements.

G. Decay Operations for Hot RRM Entries

The bottom-right portion of Figure 6 shows the RRM Decay Operations for Hot RRM Entries. In order to prevent obsolete *hot* Retention Regions (i.e., the Retention Regions that were previously frequently written but are now infrequently written) from taking unnecessary selective fast refreshes, we need a decay mechanism to detect the obsolete *hot* Retention Regions and transfer them back to *cold* status. To do this, we have a 4-bit *decay_counter* in every RRM entry. Every 0.125 seconds (1/16 of the interval between two short retention interupts), we cyclically increase the *decay_counter*. Once the *decay_counter* returns to 0, we check the value of *dirty_write_counter* as follows:

- If *dirty_write_counter* equals *hot_threshold* (16 by default), it means the Retention Region is still *hot* during the last short retention interval. In this case, we keep the RRM entry as *hot*, and halve the value of *dirty_write_counter* to check its hotness in the coming interval.
- Otherwise (*dirty_write_counter* smaller than *hot_threshold*), it indicates that the Retention Region is not *hot* enough during the last short retention interval. In this case, we first set the *hot* bit of the RRM entry back to 0. Then, for every memory block in the RRM entry that uses the short retention mode (*short_retention_vector[x]==1*), the RRM sends a slow refresh request to the memory controller. Finally, the *short_retention_vector* is reset to 0.

H. Controlling the Aggressivenes through Hot_Threshold

Hot_threshold refers to how many dirty writes an RRM entry needs to accumulate to be identified as a *hot* RRM entry (Section IV-C). It is the major attribute to control the aggressiveness of RRM:

- If this threshold is lower, then more RRM entries will be identified as *hot*, resulting in more *3-SETs-Writes* which improve the performance. However, there will also be more false positive identifications, that is, some RRM entries that are not that frequently written will be also identified as *hot*, and the system has to issue unnecessary *RRM Refresh Requests* for these entries. This, in turn, shortens the memory lifetime.
- If this threshold is higher, then some frequently accessed RRM entries cannot be identified as "hot" in time. As a result, fewer *3-SETs-Writes* will be issued, and this lowers the performance but extends the memory lifetime.

As will be shown in Section VI-D, altering *hot_threshold* can achieve fine-grained tuning of RRM's behavior. So the system users can easily configure the aggressiveness of RRM to achieve either higher performance or a longer lifetime.

Table IV
PROCESSOR CONFIGURATION.

Freq.	2GHz
# Cores	4
Core	Alpha ISA, OoO, 8 issue
L1 Caches	split 32KB I/D-cache/core, 4-way, 2-cycle hit latency, 8-MSHR
L2 Caches	256KB/core, 8-way, 12-cycle hit latency, 12-MSHR
L3 Cache (LLC)	shared 6MB, 24-way, 35-cycle hit latency, 32-MSHR
	4KB entry coverage, 24-way, 256 sets 4-cycle access latency hot_threshold: 16 LLC coverage rate: 4× storage: **96KB (1.56% of LLC)**

Table V
MLC PCM MEMORY CONFIGURATION.

Memory Size	8GB
Freq.	400 MHz
Bus Width	64-bit
# of Channels	4
# of Banks	16 per channel
RRM Refresh Queue	64 entries per channel, with high priority
Read Queue	32 entries per channel, with middle priority
Write Queue	64 entries per channel, with low priority
Page Policy	open page
Row Size	16KB
Row Buf Size	1KB
tRCD	48 cycles (120ns)
tCAS	1 cycle (2.5ns)
tFAW	50ns
tWP (write pulse time)	3-SETs-Writes: 220 cycles (550 ns); 4-SETs-Writes: 280 cycles (700 ns); 5-SETs-Writes: 340 cycles (850 ns); 6-SETs-Writes: 400 cycles (1000 ns); 7-SETs-Writes: 460 cycles (1150 ns).
Endurance	5×10^6 writes
Misc	Assuming an effective wear leveling scheme (e.g., [13]), which makes the whole memory achieve 95% of the average cell lifetime. Using Write Pausing [14] technique. Using Write-through (bypassing row buffer)

V. METHODOLOGY

We implement our proposed structure based on GEM5 [15] integrated with NVmain [16], which provides a full-system simulation with time-accuracy, non-volatile main memory technology support. Based on this platform, we simulate a CMP processor with the parameters of state-of-art Intel i5 processor; the detailed configurations are shown in Table IV. Here we implement an RRM with 4KB entry size, 24-way associativity and 256 sets. As a result, the RRM manages to cover 24MB of the memory, which is 4× of the LLC size. The *hot_threshold* of every RRM entry is 16. Such an RRM requires 96KB of the storage, which is only 1.56% of the storage required by the LLC data arrays. In Section VI, we also present sensitivity studies for various attributes of RRM (e.g., LLC coverage rate, entry coverage size, *hot_threshold*, etc.).

Table V presents the configurations of the simulated MLC PCM memory and its associated memory controller. Our PCM cells have a representative endurance of 5×10^6 writes, and we assume that an effective wear leveling scheme (such as [13]) is adopted and makes the whole PCM memory

Table VI
SIMULATED SCHEMES.

Static-7-SETs	globally using 7-SETs-Writes, global refresh in every 3054 seconds.
Static-6-SETs	globally using 6-SETs-Writes, global refresh in every 991 seconds.
Static-5-SETs	globally using 5-SETs-Writes, global refresh in every 104 seconds.
Static-4-SETs	globally using 4-SETs-Writes, global refresh in every 24 seconds.
Static-3-SETs	globally using 3-SETs-Writes, global refresh in every 2 seconds.
RRM	our proposed scheme, selectively using 3-SETs-Writes and 7-SETs-Writes, global refresh in every 3054 seconds (using 7-SETs-Writes).

Table VII
WORKLOADS AND THEIR MPKI (MISS PER 1000 INSTRUCTIONS).

Workload	MPKI	Workload	MPKI
bwave	11.69	GemsFDTD	26.56
hmmer	2.84	lbm	55.15
leslie3d	10.46	libquantum	52.07
mcf	73.42	milc	34.40
zeusmp	7.64		

MIX_1	mcf+bwave+zeusmp+milc
MIX_2	GemsFDTD+libquantum+lbm+leslie3d

achieve 95% of the average cell lifetime . We model a conventional MLC PCM memory controller, which have 4 channels with 16 banks per channel. The MLC PCM memories have built-in refresh circuits that issue global refreshes for the memory blocks. For the processor using RRM, the MLC PCM memory controller has an additional *RRM Refresh Queue* that handles the dedicated refresh requests from RRM. To ensure these refresh operations finish in time, we give *RRM Refresh Queue* the highest priority. Because there are relatively few memory blocks which need to be selectively refreshed by RRM, in our experiment, we did not encounter any situation where an RRM refresh request does not meet the retention timing request.

Table VI presents the simulated schemes. Here *Static-N-SETs* refers to the static scheme that globally uses the *N-SETs-Writes*. Different static schemes have different global refresh intervals, from 2 seconds in *Static-3-SETs* to 3054 seconds in *Static-7-SETs*. For our proposed RRM scheme, the memory controller selectively uses *3-SETs-Writes* and *7-SETs-Writes* under the directory of the RRM structure. The RRM scheme also uses global refresh with *7-SETs-Writes*. For simulation simplicity we do not simulate performance degradation caused by global refreshes. For schemes with long global refresh intervals (i.e., *Static-7-SETs*, *Static-6-SETs*, *Static-5-SETs* and *RRM*), such an approximation results in hardly any performance difference since the global refresh time is negligible compared to the global refresh intervals; however, for the schemes with relatively short global refresh intervals (i.e., *Static-4-SETs* and *Static-3-SETs*), their performance may be greatly degraded by global refresh operations. *Therefore, in the paper, the performance reported for Static-4-SETs and Static-3-SETs schemes may be significantly higher than their actual performance.* Meanwhile, for the RRM scheme, we simulate in detail the dedicated refresh operations issued by the RRM structure.

As is shown in Table VII, we use 9 memory-intensive SPEC2006 [17] benchmarks. Our experiment includes both single-benchmark and mixed-benchmark workloads. A single-benchmark workload has 4 identical copies of a single benchmark, and a mixed-benchmark workload (i.e., `MIX_1` or `MIX_2`) has 4 different benchmarks. We run each workload for 5 seconds.

VI. RESULTS

In order to evaluate the effectiveness of RRM, we provide a series of comparison results between RRM and various static write schemes. The evaluation metrics include performance, lifetime, wear-out distribution, energy consumption, etc. Then we provide a sensitivity study of different RRM design choices.

A. Performance vs. Lifetime

As shown in Figure 7, RRM provides a significant performance improvement over all *Static-7-SETs* schemes with a geometric mean of 62.0% and up to 64.1% higher IPC with `Mix_2`. Although it cannot achieve the same performance with *Static-3-SETs*, it outperforms the second high-performance static scheme *Static-4-SETs*. In particular, as mentioned in Section V, the actual performance of *Static-3-SETs* will be considerably worse, so the performance gap between *Static-3-SETs* and *RRM* is actually smaller.

Figure 8 shows that *RRM* achieves a remarkable lifetime improvement compared to *Static-3-SETs* and *Static-4-SETs* schemes. This is because *RRM* only issues relatively small amounts of RRM refresh requests to *hot* RRM regions as opposed to the latter two static schemes, which need to globally issue refresh requests to every memory block. Meanwhile, the *Static-7-SETs* scheme still achieves a geometric mean of 66.3% longer lifetime than *RRM*. The main reason for *RRM*'s shorter lifetime is that *RRM* considerably outperforms *Static-7-SETs*—as a result, there are more writes issued (thus more wear is introduced) by *RRM* within the same amount of time. Although *RRM* also selectively issues *RRM Refresh Requests*, the wear introduced is not the dominant factor of lifetime, as will be shown in Section VI-B.

B. Wear Distribution

Figure 9 shows the wear of different schemes within the same amount of time (in our case, 5 seconds). We also differentiate the wear introduced by write operations from the wear introduced by refresh operations.

We can see that, from *Static-7-SETs* to *Static-3-SETs*, as the number of SETs used in write operations decreases, the wear introduced by refresh operations becomes more significant. For *Static-3-SETs* and *Static-4-SETs*, the wear of refresh operations even becomes the dominant factor.

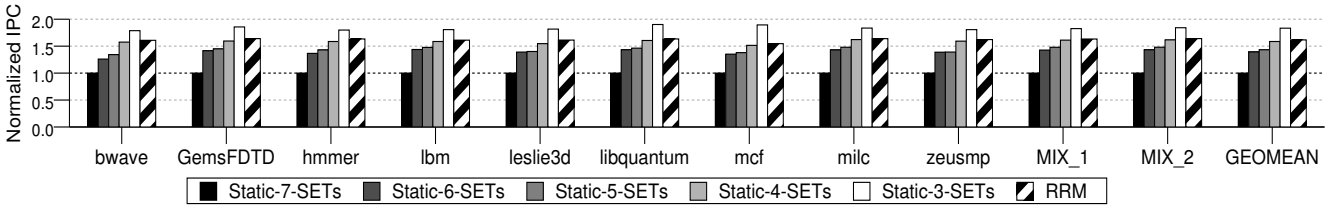

Figure 7. **Normalized IPC comparison.**

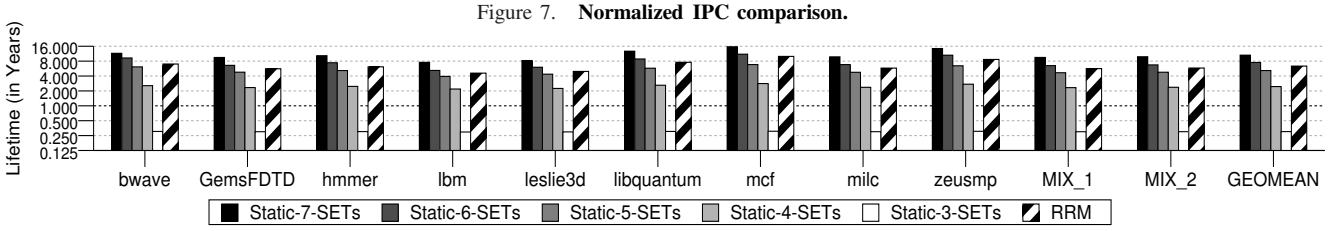

Figure 8. **Lifetime comparison.**

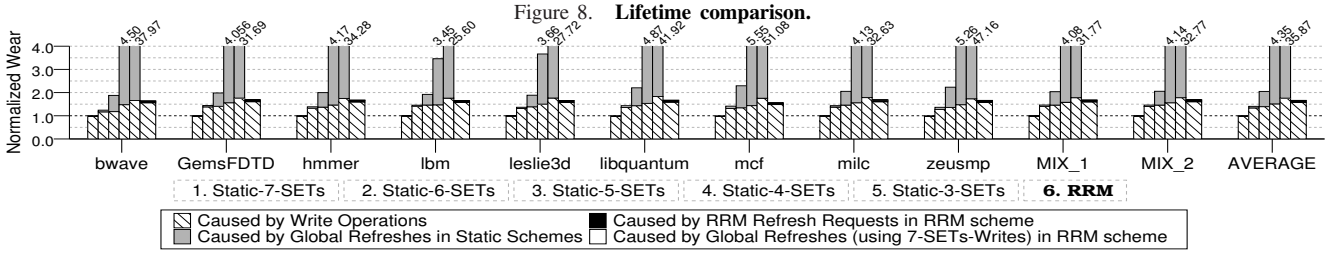

Figure 9. **Wear distribution within the same amount of time.**

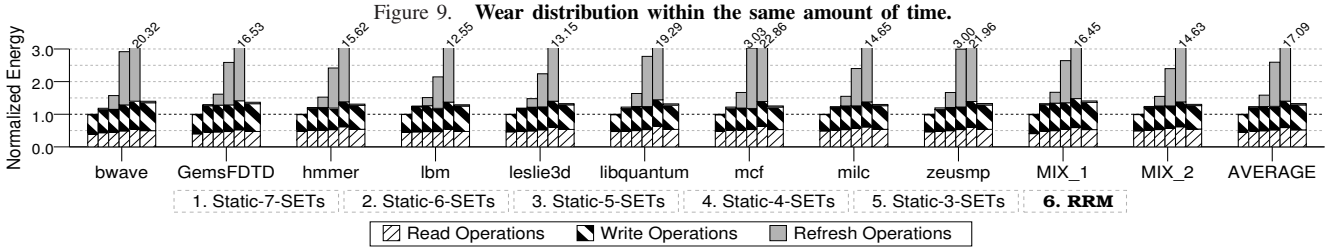

Figure 10. **Memory energy consumption.**

In *RRM*, there are two types of refresh operations: *RRM Refresh* operations issued by the RRM structure and *Global Refresh* operations, which are the same as those for *Static-7-SETs*. However, we can see that both types of refreshes introduce a fairly limited amount of wear compared to the wear introduced by the write operations. This indicates that RRM does a good job in identifying and refreshing the *hot* memory region that is limited in size.

C. Memory Energy Consumption

We calculate the PCM energy consumption with the energy parameters listed in Table I. As is shown in Figure 10, for static schemes the energy consumed by refresh operations increases dramatically as the number of SETs in a write decreases as a result of decreasing refreshing intervals. For *Static-3-SETs* and *Static-4-SETs*, the refreshing energy is dominant among all the memory energy costs. This means these two schemes are also impractical from an energy/power point of view.

On the other hand, our proposed *RRM* scheme has a trivial amount of energy consumed by refresh operations, and its overall energy consumption is moderate—32.8% more than the energy of *Static-7-SETs*. As is discussed in Section VI-B, the extra energy is mainly caused by the fact that *RRM* outperforms the *Static-7-SETs* and thus issues more memory operations.

D. Aggressiveness Control

As is discussed in Section IV-H, we can control the aggressiveness of RRM through *hot_threshold*. Here we examine the system behavior under four *hot_threshold* values (i.e., 8, 16, 32 and 64) to check how this threshold affect the system performance and memory lifetime. As shown in Figure 11, as we expect, the performance goes down and the lifetime goes up as threshold increases. Given the performance and lifetime benefits presented in Section VI-A, we believe a threshold of 16 is a decent choice. However, if some system users want a system with higher performance and do not care about memory lifetime that much, they can choose a *hot_threshold* of 8. This configuration can achieve a system performance which is 9.0% higher than the default performance with a *hot_threshold* of 16 and still has a lifetime of 5.78 years. Note that, such a system is only 3.6% inferior in performance than *Static-3-SETs*. On the hand, if

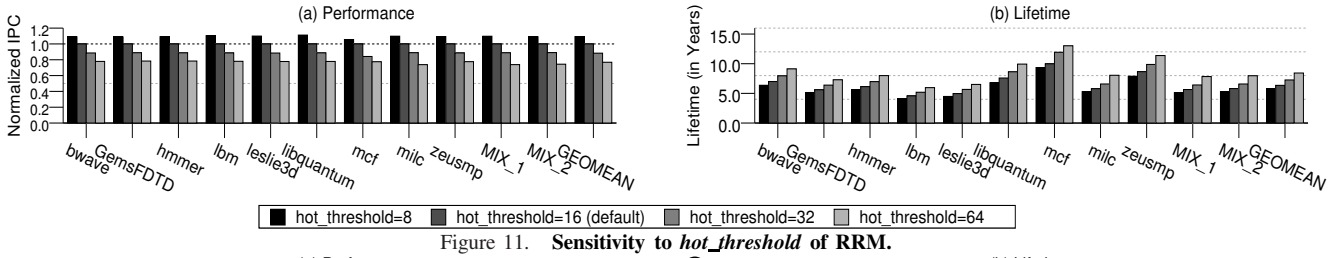

Figure 11. **Sensitivity to *hot_threshold* of RRM.**

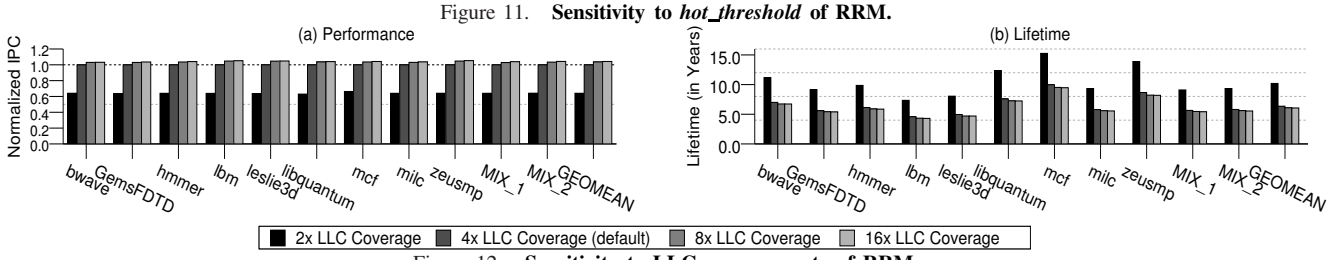

Figure 12. **Sensitivity to LLC coverage rate of RRM.**

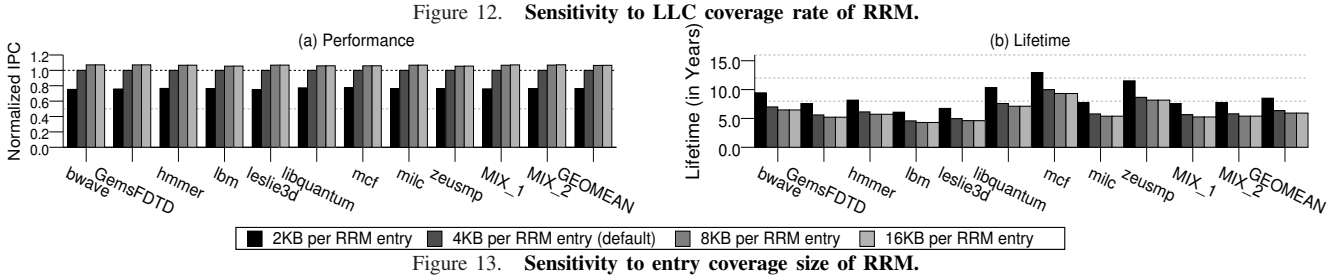

Figure 13. **Sensitivity to entry coverage size of RRM.**

Table VIII
RRM CONFIGURATION FOR DIFFERENT LLC COVERAGE.

LLC Coverage	Configuration	Overhead
2×	128 sets, 24 ways	48KB, 0.78% of LLC
4× (default)	**256 sets, 24 ways**	**96KB, 1.56% of LLC**
8×	512 sets, 24 ways	192KB, 3.12% of LLC
16×	1024 sets, 24 ways	384KB, 6.25% of LLC

a system user wants longer memory lifetime, he/she can set a higher value (e.g., 32 or 64) to *hot_threshold*, though such configuration will result in lower system performance.

E. Sensitivity to LLC Coverage Rate of RRM

The LLC coverage rate of *RRM* refers to the ratio of the amount of memory covered by *RRM* to the amount of memory covered by LLC. A higher LLC coverage rate of *RRM* means RRM will be more effective since more RRM entries will be tracked. However, it also involves a higher hardware overhead for *RRM*. Here we fix other attributes of *RRM* and just vary its number of sets to get multiple *RRM* schemes with different coverage rates.

We can see from Figure 12 that, an *RRM* of $2\times$ coverage has considerably worse performance (with a geometric mean of 64.0%) than those with a $4\times$ coverage *RRM*. This is because there are frequent contentions among the entries with $2\times$ coverage *RRM* and so more *RRM* replacements. As a result, there will be fewer *hot* RRM entries in the RRM structure, resulting in fewer *3-SETs-Writes*, and this lowers performance.

On the other hand, increasing the coverage rate from $4\times$ to $8\times$ and $16\times$ makes hardly any performance or lifetime difference. This means a $4\times$ coverage rate is already enough, and there is no need to further increase it at the expense of more hardware cost. Note that, a $4\times$ coverage *RRM* only requires 1.56% of the data storage of LLC (as shown in Table VIII), which is a trivial overhead compared to its lifetime/performance benefits. Therefore, our default coverage choice of $4\times$ is a sweet spot among performance, lifetime and hardware overhead.

F. Sensitivity to Entry Coverage Size of RRM

In *RRM*, entry coverage size refers to the memory range an *RRM* entry covers. In our default setting, entry coverage size is set as 4KB. To check the effectiveness of our choice, for an *RRM* structure with $4\times$ coverage, we vary this attribute from 2KB to 16KB. Figure 13 shows the corresponding performances and lifetimes.

We can see that the *RRM* with a 2KB entry coverage size performs considerably worse than other design choices. This is because the smaller entry coverage size makes it harder for the *RRM* entries to accumulate enough dirty writes to be identified as "hot". As a result, the *RRM* issues fewer *3-SETs-Writes*.

On the other hand, the *RRMs* with 4KB/8KB/16KB entry coverage sizes have similar performance/lifetime. We use a default entry coverage size of 4KB because most commercial computing systems (e.g., the ones with x86_64 CPUs) usually have the same page size. If an RRM entry covers more than 4KB (e.g., 8KB or 16KB), it may cover discontinuous logical pages with different memory access behavior, and this situation may lower the accuracy of *RRM*.

VII. Related Work

We presented a detailed discussion of the proposals that use the same write-latency vs. retention trade-offs in Section III-B. In this section, we present other related work. Non-volatile memory technologies (e.g., PCM [18], [19] and RRAM [20], [21]) are considered as potential alternative solutions for main memory [22], [23], [24] and persistent storage [25], [26]. However, limited write endurance and unavoidable long write latency make it impossible for NVM to directly replace the traditional DRAM memory and hard disk. To counter these two problems, various solutions have been proposed.

For the write endurance issue, the most straightforward approaches are to limit the wear of the writes [22], [23], [24], [27], [21]. There are also various wear-leveling proposals [13], [28], [29], [24] that focus on avoiding the write hotspots in the NVM memories. Another way to prolong the NVM lifetime is to keep the NVM functional when it is partially worn out. This kind of solutions is usually implemented with error correction schemes [30], [31], [32].

For the performance issue, several proposed solutions are based on enhancing the parallelism of write access to hide the write latency [33], [34], [35]. Another way is to have a large cache (e.g., DRAM Caches) in front of the NVM [18], [19]. Qureshi et al. [14] propose to cancel or pause the on-going write operations when critical read requests arrive which improves the lifetime. Qureshi et al. [14] also explore the latency differences between fast RESET and slow SET operations and propose a scheme to improve the write performance by proactively performing the SET operations. Also, some proposals [36], [37], [38], [27] explore the trade-off between storage density and write/read latency by adaptively adopting SLC and MLC memory access operations.

VIII. Conclusions

In MLC PCM, there exists a trade-off between data retention and write latency That is, a longer MLC PCM write can have better resistance precision and thus achieve better data retention time because it is more tolerant to resistance drift. However, just adopting a single write type is not ideal. On one hand, if just long-latency-long-retention writes are used, the system performance suffers because of the long write latency; on the other hand, if just short-latency-short-retention writes are used, the memory has an unacceptably short lifetime because of the frequent global memory refreshes.

In this paper, we show that the memory write operations are highly imbalanced: a relatively small number of the memory regions (i.e., *hot memory region*) get most of the writes. Based on such an observation, we propose *Region Retention Monitor (RRM)*, a novel structure that can automatically identify the *hotness* of memory blocks and assign a proper type for the current write operation—if the write is to a *hot* memory block, then it should be a short-latency-short-retention write; otherwise, it should be a default long-latency-long-retention one. *RRM* also issues selective memory refreshes to ensure the correctness of the memory data.

The experiment results show that, as a geometric mean, a system with *RRM* outperforms a system statically using long-latency-long-retention writes (*7-SETs-Writes*) by 61.0%; and its performance is only 10.0% inferior to a system statically using short-latency-short-retention writes (*3-SETs-Writes*). As a geometric mean, a system with *RRM* achieves 6.4 of years memory lifetime; for comparison, systems statically using *7-SETs-Writes* and *3-SETs-Writes* achieve 10.6 and 0.3 years of memory lifetime, respectively. Such results lead us to conclude that *RRM* is effective in balancing system performance and memory lifetime. In the meantime, a more aggressively configured *RRM* can achieve a performance which is only 3.5% inferior to a system statically using short-latency-short-retention writes (*3-SETs-Writes*) while still achieves a lifetime of 5.78 years.

IX. Acknowledgements

We thank the anonymous reviewers for their insightful comments and suggestions. This work is supported in part by NSFC grants No. 61520106005 and 61521092. Mingzhe Zhang is supported by China Scholarship Council under Grant No. 201504910535. We acknowledge support from the University of Chicago Research Computing Center.

References

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

[2] Q. Li, L. Jiang, Y. Zhang, Y. He, and C. J. Xue, "Compiler directed write-mode selection for high performance low power volatile pcm," in *LCTES*, 2013.

[3] C.-M. Jung, E.-S. Lee, K.-S. Min, and S.-M. S. Kang, "Compact verilog-a model of phase-change ram transient behaviors for multi-level applications," vol. 25, no. 7, 2011.

[4] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *MICRO*, 2009.

[5] K. Kim and S. J. Ahn, "Reliability investigations for manufacturable high density pram," in *IRPS*, 2005.

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

[7] Q. Li, L. Jiang, Y. Zhang, Y. He, and C. J. Xue, "Compiler directed write-mode selection for high performance low power volatile pcm," in *ACM SIGPLAN Notices*, vol. 48, no. 5. ACM, 2013, pp. 101–110.

[8] K. Qiu, Q. Li, and C. J. Xue, "Write mode aware loop tiling for high performance low power volatile pcm," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–6.

[9] C. Pan, M. Xie, J. Hu, Y. Chen, and C. Yang, "3m-pcm: exploiting multiple write modes mlc phase change main memory in embedded systems," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2014, p. 33.

[10] D. Kang, S. Baek, J. Choi, D. Lee, S. H. Noh, and O. Mutlu, "Amnesic cache management for non-volatile memory," in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–13.

[11] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang, "Nvm duet: Unified working memory and persistent store architecture," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 455–470, 2014.

[12] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing nand flash-based ssds via retention relaxation," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208461.2208472

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

[14] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–11.

[15] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[16] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, 2012, pp. 392–397.

[17] "Spec2006 benchmarks," http://www.spec.org/cpu2006/.

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

[19] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009.

[20] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 476–488.

[21] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs." ISCA, 2016.

[22] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 2–13.

[23] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[24] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH computer architecture news*, vol. 37, no. 3. ACM, 2009, pp. 14–23.

[25] A. M. Caulfield, T. I. Mollov, L. A. Eisner, A. De, J. Coburn, and S. Swanson, "Providing safe, user space access to fast, solid state disks," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 387–400, 2012.

[26] I. Moraru, D. G. Andersen, M. Kaminsky, N. Tolia, P. Ranganathan, and N. Binkert, "Consistent, durable, and safe memory management for byte-addressable non volatile main memory," in *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems*. ACM, 2013, p. 1.

[27] L. Zhang, D. Strukov, H. Saadeldeen, D. Fan, M. Zhang, and D. Franklin, "Spongedirectory: Flexible sparse directories utilizing multi-level memristors," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, 2014, pp. 61–74.

[28] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure pcm systems by online detection of malicious write streams," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 478–489.

[29] 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," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 383–394, 2010.

[30] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie memory: Extending memory lifetime by reviving dead blocks," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 452–463.

[31] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 433–444.

[32] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 318–328.

[33] S. Cho and H. Lee, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2009, pp. 347–357.

[34] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing pcm banks from seizing too much power," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 186–195.

[35] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "Fpb: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 1–12.

[36] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 153–162.

[37] Z. Deng, L. Zhang, D. Franklin, and F. T. Chong, "Herniated hash tables: Exploiting multi-level phase change memory for in-place data expansion," in *Proceedings of the 2015 International Symposium on Memory Systems*, 2015, pp. 247–257.

[38] X. Dong and Y. Xie, "Adams: Adaptive mlc/slc phase-change memory design for file storage," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, 2011, pp. 31–36.