WALL: A Writeback-Aware LLC Management for PCM-based Main Memory Systems

Bahareh Pourshirazi[†], Majed Valad Beigi[‡], Zhichun Zhu[†] and Gokhan Memik[‡]
[†]Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, USA
[‡]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, USA
bpours2@uic.edu, majed.beigi@northwestern.edu, zzhu@uic.edu and memik@eecs.northwestern.edu

Abstract— **In this paper, we propose WALL, a novel writeback-aware LLC management scheme to reduce the number of LLC writebacks and consequently improve performance, energy efficiency, and lifetime of a PCM-based main memory system. First, we investigate the writeback behavior of LLC sets and show that writebacks are not uniformly distributed among sets; some sets observe much higher writeback rates than others. We then propose a writeback-aware set-balancing mechanism, which employs the underutilized LLC sets with few writebacks as an auxiliary storage for storing the evicted dirty lines of sets with frequent writebacks. We also propose a simple and effective writeback-aware replacement policy to avoid the eviction of the writeback blocks that are highly reused after being evicted from the cache. Our experimental results show that WALL achieves an average of 26.6% reduction in the total number of LLC writebacks, compared to the baseline scheme, which uses the LRU replacement policy. As a result, WALL can reduce the memory energy consumption by 19.2% and enhance PCM lifetime by 1.25×, on average, on an 8-core system with a 4GB PCM main memory, running memory intensive applications.**

Keywords: Phace Change Memory; Write Endurance; Energy Consumption; Performance; Last Level Cache

I. Introduction

DRAM can no longer satisfy the memory capacity demands of the modern-day applications due to its scalability limit; it is difficult to scale DRAM cells down to feature sizes smaller than 16nm [20]. Moreover, DRAM consumes considerable amount of static and refresh power. It is estimated that DRAM power constitutes to 20% to 40% of the total system's energy in today servers [10]. Phase Change Memory (PCM) is a promising technology that is gaining interest as a DRAM replacement for building the future main memories [4, 12-14, 16]. PCM is a random-access memory that is about 2 to 4 times denser than DRAM [9]. In addition to scalability, zero standby power, low read latencies, and resiliency to soft errors are some of the other intriguing features of a PCM-based main memory [16]. On the other hand, PCM suffers from some major shortcomings including long write latency, high write energy consumption, and limited write endurance, which are all related to write operations. The impact of write operations on PCM is discussed in detail in Section II. A.

To deal with the above-mentioned overheads of the write operations in PCM, there are two common types of solutions. First category is the optimizations on the PCM architecture to minimize the impact of writes. For example, modifying the request scheduling policy in the PCM main memory to alleviate the performance overheads of writes on reads [3, 15, 19, 23]; or modifying the PCM main memory architecture to reduce/balance the write load on PCM cells and enhance their lifetime [22]. Second category is reducing the total number of writes sent to the PCM main memory by modifying the Last Level Cache (LLC)'s management policies [18]. In this study, we focus on the latter.

In this paper, we propose ***WALL***, a novel **W**riteback-**A**ware **LL**C management scheme to improve performance, energy efficiency, and lifetime of a PCM-based main memory system by reducing the number of writebacks from LLC to PCM. In general, WALL consists of a writeback-aware set balancing mechanism and a writeback-aware replacement policy. Writebacks of the last level cache are not uniformly distributed among its sets; some sets have far more writebacks than others while some sets rarely see a writeback (see Section II. B). The proposed set balancing mechanism reduces the number of writebacks by employing the underutilized sets with infrequent writebacks as storage units (inside LLC) for storing the evicted dirty lines of sets with many writebacks. Moreover, the proposed writeback-aware replacement policy tries to keep the dirty blocks that are frequently accessed after eviction in LLC. To do so, it allows the dirty eviction victims (i.e., dirty LRU block) to stay in the cache and be re-accessed; if the block becomes LRU block again without being accessed, it will be evicted from LLC then.

The rest of this paper is organized as follows. Section II discusses background and motivation. Section III describes the proposed scheme. Section IV presents the experimental results. The related work is discussed in Section V. Finally, Section VI summarizes the work.

II. Background and Motivation

A. *The Impact of Write Operations on PCM*

Phase Change Memory (PCM) is a type of non-volatile memory technology and has been extensively explored as a DRAM alternative. The main advantages of PCM over DRAM are its scalability and non-volatility. However, PCM has higher write access energy and latency (2 to 8 times [9]) and limited write endurance. A previous study [3] has shown that the long-latency write operations can increase the effective latency of read requests by 1.2 to 1.8 times. In addition, the number of write operations in PCM affects its lifetime. The PCM-based

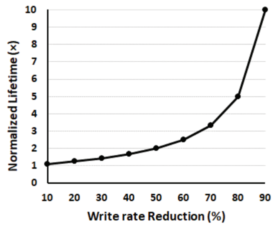

Figure 1. Impact of reducing write traffic on PCM lifetime.

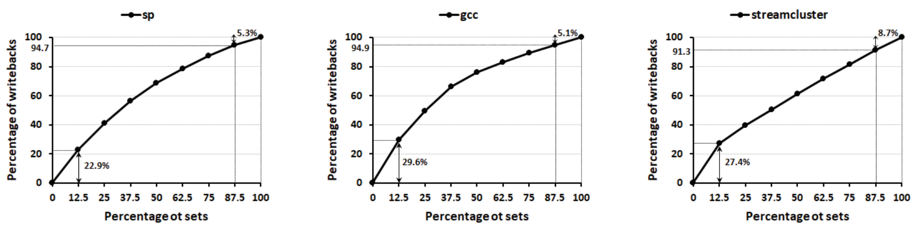

Figure 2. Cumulative distribution of writebacks over LLC sets. (*NOTE*: sets are sorted in a descending order based on their total number of writebacks)

memory system lifetime can be estimated as [14]:

$$\boldsymbol{System\ Lifetime} = \boldsymbol{Y}\ (\boldsymbol{years}) \approx \frac{\boldsymbol{w_{max}} \cdot \boldsymbol{S}}{\boldsymbol{B}} \cdot \boldsymbol{2^{-25}}$$

In this formula, $\boldsymbol{B}$ is the write traffic (or write rate, GBps), $\boldsymbol{Y}$ is the maximum number of years that a PCM with size $\boldsymbol{S}$ and cell endurance of $\boldsymbol{w_{max}}$ can last. The effect of reducing the write traffic on the expected PCM lifetime is shown in Figure 1. It should be noted that the results of the figure are independent of S and w_{max}. We also assume that the writes are distributed uniformly across the entire PCM memory.

B. Motivation

In this section, we explain the motivation of our work by investigating LLC sets' writeback behavior. To this end, we have run three workloads, selected from different benchmark suites (*sp* from NAS [2], *gcc* from SPEC CPU2006, and *streamcluster* from PARSEC [5]) on our simulated system (see Section IV for more details). Figure 2 shows the cumulative distribution of writebacks over LLC sets.

The results reveal that majority of the writebacks from the LLC to main memory are generated by less than half of the LLC sets. For *sp*, 25% of the LLC sets, with the largest number of writebacks (i.e., *frequent writeback sets*), are responsible for 41.1% of the total number of writebacks. On the other hand, 12.5% of the sets, with the smallest number of writebacks (i.e., *infrequent writeback sets*), are accountable for only 5.3% of the total number of writebacks. In general, some sets tend to write back much more frequently than others. For example, since 22.9% of the writebacks for *sp* are performed by 12.5% of the *frequent writeback sets*, a *frequent writeback set* writes to main memory about four times more frequently than an *infrequent writeback set*, on average. Similarly, for *gcc*, 25% of the *frequent writeback sets* perform 49.6% of the writebacks while the same percentage of the *infrequent writeback sets* are responsible for only 10.8% of the total number of writebacks. Finally, for *streamcluster*, 20.7% of the writebacks are sourced from 6.25% of the *frequent writeback sets*.

Based on this observation, we propose a set balancing mechanism to reduce the number of writes to a PCM-based main memory system. More specifically, WALL aims to prevent a noticeable percentage of write traffic from reaching the PCM main memory by taking the non-uniform distribution of LLC set writebacks into account. The idea of avoiding the eviction of LLC sets' highly reused dirty blocks, which are likely to become eviction victims soon after being re-inserted in the cache (i.e., *frequent writeback blocks*), has been discussed in a recent study called *WADE* [18]. WADE partitions the blocks of each LLC set into two groups, *frequent writeback blocks* and *nonfrequent writeback blocks* and tries to keep the *frequent writeback blocks* in the set to reduce the number of writes to the main memory. However, one major shortcoming of WADE is that for every set, irrespective of its characteristics, it considers the *nonfrequent writeback blocks* of the set as the only replacement candidates; while other sets might have clean, underutilized lines. Moreover, WADE is rather complex. In this study, a simple but effective writeback-aware replacement policy is also proposed to keep the *frequent writeback blocks* of sets in the cache. Overall, WALL is able to reduce the number of LLC writebacks by considering the writeback behavior of sets and their blocks at the same time. The WALL scheme will be discussed next.

III. WALL Scheme

Our proposed scheme, WALL, consists of a writeback-aware set balancing mechanism and a writeback-aware replacement policy. The WALL set balancing mechanism classifies LLC sets into three cathegories, ***writers*** (i.e., *frequent writeback sets*), ***non-writers*** (i.e., *infrequent writeback sets*) and ***neutral***. To reduce the number of write requests to PCM, the *non-writer* sets are used to store the evicted dirty lines of the *writer* sets. Specifically, each *writer* set is partnered with a *non-writer* set (until there is no *non-writer* set left) and upon eviction of a LRU dirty line from the *writer* set, the line will be inserted into the set's partner instead of being written back to the main memory. Besides, our writeback-aware replacement policy further reduces the number of LLC writebacks by keeping the *frequent writeback blocks* in the cache.

A. Writeback-Aware Set Balancing LLC

To decide whether a set is a *writer* or *non-writer*, WALL monitors the number of writebacks and accesses of each LLC set for a period. Generally, a set is "*writer*" if the number of writebacks from it exceeds a certain threshold (i.e., $\boldsymbol{\tau_{high\_wb}}$), but a set with relatively smaller number of writebacks is not necessarily a good partner set. A *non-writer* set must have enough space to store the evicted dirty blocks of its *writer* partner without noticeable performance penalty. To measure the degree to which a set can hold its working set, we employ a saturating arithmetic miss counter (i.e., saturation counter) similar to [17]. For a K-way set associative cache, the working range of the saturation miss counter is from 0 to 2K-1 [17]. Upon every access, the saturation counter is incremented if the access results in a miss, and decremented otherwise. To count the number of writebacks, we also use a writeback frequency counter for each set that is incremented upon every writeback from the set.

WALL considers a set "*non-writer*" if both of the set's saturation and writeback counters are smaller than specific thresholds (i.e., $\boldsymbol{\tau_{sat}}$, $\boldsymbol{\tau_{low\_wb}}$). For a set with writeback frequency counter of W and saturation counter of M, the set is considered *writer* if ($W \geq \tau_{high\_wb}$), *non-writer* if ($M \leq \tau_{sat}$ & $W \leq \tau_{low\_wb}$) and *neutral* otherwise. We divide the total execution time of programs into epochs of 10^7 accesses to the LLC. On entering an epoch, the partnership between two sets can be easily broken if the *writer* set has no blocks in its *non-writer* partner; then the thresholds are re-calculated and if there is a pair of *writer* and *non-writer* sets with no partners, they will be assigned to each other. Note that for a set with partner (i.e., a *writer* set with blocks in its partner or a *non-writer* set that holds some of its partners' blocks), we do not change the set type. Our experiments show that set types rarely change after an initial set type identification epoch and most partnerships remain intact. On the other hand, to avoid overfill a *non-writer* set, if the saturation counter of a *non-writer* set reaches $\alpha \times \tau_{sat}$, where α is 2 in our experiments, the insertion of blocks from its *writer* partner will be suspended until the set's saturation counter retrieves to values smaller than τ_{sat}. To reduce implementation complexities, we assume each *writer* set is partnered with only one *non-writer* set. In other words, the number of the paired *writer* and *non-writer* sets is equal to the size of the smaller group of *writer* and *non-writer* sets. We later discuss how accesses to sets and their partners are handled in Section III. C.

To determine the writeback thresholds, three simple steps are followed. First, the arithmetic mean of all the writeback values is computed and referred to as the overall average. Then, τ_{low_wb} is computed as the arithmetic mean of the writeback values smaller than the overall average. Finally, τ_{high_wb} is computed as the arithmetic mean of the writeback values larger than the overall average. Note that since these thresholds cannot guarantee an equal number of *writer* and *non-writer* sets, some *writer* sets may remain without a partner at the end. Moreover, we assume τ_{sat} = K/4 (i.e., K is the set associativity). Figure 3 depicts the distribution of sets based on the discussed thresholds for the workloads shown in Table III (see Section IV. A). The results are based on the values obtained after the initial epoch of 10^7 accesses to the LLC. It should be noted that the saturation bars (i.e., bars with horizontal axis title of *sat*) in the figure are only for the sets with writeback counters smaller than τ_{low_wb}. Our results show that the selected thresholds can distinguish different types of sets from each other effectively.

B. Writeback-Aware Replacement Policy

In addition to assigning partners to the *writer* sets, WALL enables a writeback-aware replacement policy inside the LLC sets to further reduce the number of writebacks to the PCM. To keep the *frequent writeback blocks* in the cache, such blocks need to be identified first. We propose a much simpler yet effective method compared to the prediction scheme discussed in WADE [18], because such prediction schemes are usually complex and costly in terms of area, energy, and/or performance overheads.

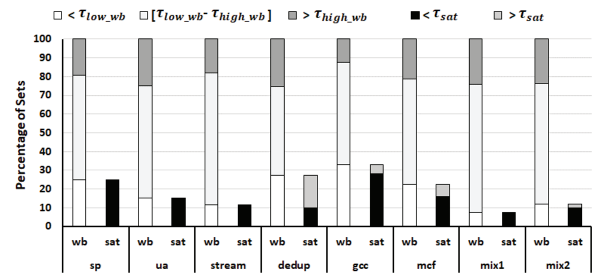

Figure 3. Distribution of LLC sets based on the thresholds (wb: writebacks; sat: saturation; *NOTE*: *sat* bars are only for the sets with wb < τ_{low_wb}).

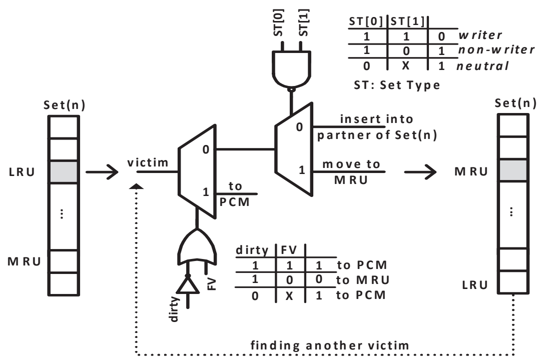

Figure 4. WALL design.

The intrinsic definition of a *frequent writeback block* is a block that is frequently reused each time after being evicted from the cache. Generally, our scheme avoids the eviction of such blocks by giving the dirty victims a second chance to stay in the cache and be accessed again. To keep track of the dirty blocks that have been given a second chance, a one-bit flag called ***FV*** (i.e., Former Victim) is considered for each block. We assume the baseline replacement policy is LRU. When a replacement is needed in a LLC set, the dirty status bit of the LRU line (i.e., the eviction victim) is checked; if the LRU block is clean, it will be evicted from the cache but if the block is dirty, two scenarios are possible. First, the block is not a former victim (*FV* = 0). In this case, the line will be moved to the MRU position of the access stack and will be marked as former victim (i.e., its *FV* flag will be set to '1'), this process will be repeated until finding an eviction victim. Second, the block is a former victim (*FV* = 1) and has become the eviction victim for the second time without being accessed. In this case, the block will be evicted from the cache. If a cache line with *FV* = 1 is accessed, its *FV* bit will be reset to '0'. The reason is that such block is likely to be a *frequent writeback block*. It should be noted that all these steps happen in parallel with the resolution of the miss, thereby there is no performance penalty.

We use the proposed replacement policy for the *non-writer* and *neutral* sets. Since *writer* sets usually have high miss rate values and often choose dirty blocks as eviction victims, for those sets, we use the baseline LRU replacement policy. However, on eviction of a LRU dirty block from a *writer* set with a partner, the block will be inserted into the set's partner.

TABLE I. TOTAL STORAGE OVERHEAD.

Type	Storage	Type	Storage
FV per block	16 KB	Saturation counter (6-bit) per set	3 KB
RB per block	16 KB	Writeback counter (8-bit) per set	4 KB
P per set	0.5 KB	Remap table	6 KB
ST per set	1 KB	**TOTAL**	**46.5 KB**

TABLE II. SYSTEM CONFIGURATION.

Processor and On-chip Caches	
Cores	8 cores, out-of-order, 2.0 GHz
L1-I/D	Split 32KB I/D-cache/core, 4-way, 8-MSHR, 2-cycle hit
L2	256KB/core, 8-way, 12-MSHR, 12-cycle hit
L3 (LLC)	Shared, 8MB, 32-way, 32-MSHR, 35-cycle hit
Coherency	MOESI directory, 2×4 grid packet NoC, XY routing
Main Memory	
PCM	4GB, 4 Channel, 1 rank/channel, 4 banks/rank, 400 MHz t_{SET}= 150ns, t_{RESET}= 100ns, t_{RCD}= 120ns Cell endurance = 32×10^6 writes
MC	Four controllers, Open page, 32-entry queues (one read and one write queue), Write drain threshold: high = 80%, low = 50%, Address mapping: page interleaving

TABLE III. EVALUATED WORKLOADS CHARACTERISTICS. (RPKI/WPKI: MAIN MEMORY READS/WRITES PER 1000 INSTRUCTIONS)

Workload	RPKI	WPKI	Workload	RPKI	WPKI
from the NAS benchmarks (*8-Thread*)					
sp	4.98	2.55	ua	3.12	2.67
from the PARSEC benchmarks (*8-Thread*)					
stream	24.4	0.21	dedup	11.5	8.32
from the SPEC CPU2006 benchmarks					
8×gcc	7.42	1.59	8×mcf	43.5	9.02
mix1	2.91	1.90	mix2	12.7	4.21
mix1: 4×lbm, 4×bzip			mix2: 4×cactusADM, 4×leslie3D		

The design of the proposed writeback-aware replacement policy is depicted in Figure 4. On an eviction of a block from a LLC set, we first decide whether the block needs to be written back to the PCM or not. If not, depending on the set type, the block either remains in the set as MRU or will be written back into the set's partner. To specify the type of each set, a 2-bit register called ST is considered per set (i.e., the set is *writer* if ST = "11", *non-writer* if ST = "10", and *neutral* if ST = "00" or "01").

C. *Set Balancing LLC Access Management*

For *writer* sets with large number of writebacks, changing the replacement policy may cause a non-trivial increase in the sets' miss rates. Hence, the writeback-aware replacement policy is applied to those sets that are not *writer* (i.e., *neutral* and *non-writer* sets). Instead, WALL virtually increases the associativity of a *writer* set by assigning a *non-writer* partner to it. The reason that we have excluded the *neutral* sets from the partnering process is that it is not beneficial to write from one set to another set with similar writeback or miss frequencies. The partner of a *writer* set is selected randomly from the *non-writer* sets. The indices of the sets' partners are saved in a small direct-access *remap table*, which is indexed by set indices. For a set with no partner, its own index is stored. When a LRU dirty line is evicted from a *writer* set with a partner, it will be inserted into the set's partner. To show whether a block in a *non-writer* set is repositioned from the set's *writer* partner or not, a one-bit flag called ***RB*** (i.e., Repositioned Block) is considered for each block. The *remap table* is also augmented with one-bit flag ***P*** (i.e., Partnered) for each set to show whether a *writer* set has any blocks in its partner or not. Upon an access to a *writer* set, if the access results in a miss, the remap table is checked, if *P* is '1', the set's partner will be accessed for the block; otherwise, main memory will be accessed as usual. If the access also misses in the set's partner, main memory must be accessed. If all the repositioned blocks of a *writer* set get evicted from its partner, *P* flag of the *writer* set will be reset to '0'.

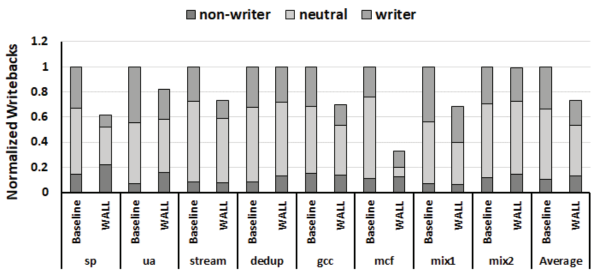

Figure 5. WALL's normalized LLC writebacks reduction.

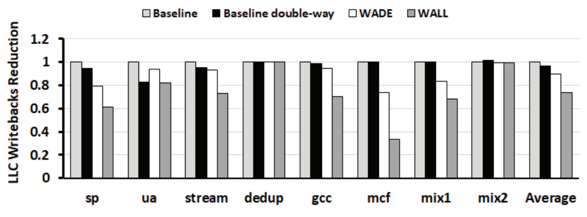

Figure 6. Writebacks reduction of evaluated schemes.

D. *Storage Overhead*

Table I summarizes the storage overhead of WALL. The total storage overhead of WALL is less than 0.6% of the LLC capacity. It is worth mentioning that this overhead is about half of that of WADE [18].

IV. RESULTS

A. *Methodology*

In this work, we model an 8-core processor using the gem5 full-system simulator [13] integrated with NVMAIN [11]. The system configuration of our experiments is shown in Table II. The PCM configurations are generated by NVSIM [8] and CACTI [1], the cell parameters used in NVSIM are based on the projections by [7]. The benchmarks used in this study are chosen from NAS [2], SPEC CPU2006 and PARSEC [5] as depicted in Table III. For all the workloads, we use either sampled reference or native input sets to represent a real-world execution scenario and run the applications for two billion instructions, after two billion for cache warm-up phase. In this paper, we compare WALL with 1) *Baseline* that uses the LRU replacement policy, 2) *Baseline double-way*, a baseline cache of the same size with double the associativity, and 3) *WADE*, which is the scheme proposed in [18]. In PCM, we always prioritize reads over writes if write queue is less than 80% full.

B. *LLC Writeback Reduction*

Figure 5 shows the writeback reduction of WALL for the different types of LLC sets. The results are normalized to the baseline scheme. It is worth mentioning that any writeback of a

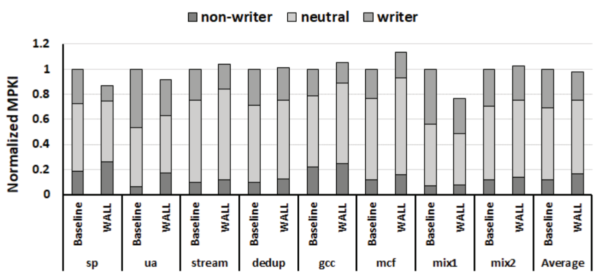

Figure 7. WALL's normalized MPKI.

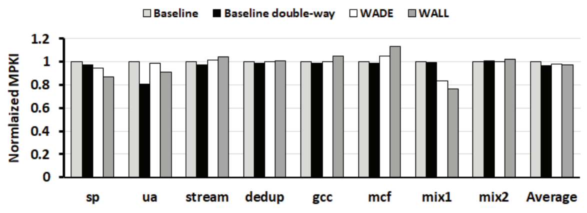

Figure 8. MPKI of evaluated schemes.

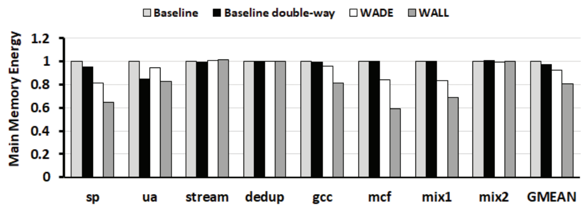

Figure 9. Normalized main memory energy.

writer set's lines from its partner is considered for the original *writer* set.

WALL achieves an average of 26.6% writeback reduction, compared to the *Baseline* scheme. The three main reasons for this reduction are: 1) The efficacy of the set type identification process. 2) The capability of the writeback-aware set balancing scheme in reducing the *writer* sets' writebacks; the number of writebacks originated from *writer* sets is reduced by 39.5%, on average (i.e., from 33.4% to 20.1% of the *Baseline* total writebacks), while the writebacks originated from *non-writer* sets is increased slightly from 10.4% to 13.1% of the *Baseline* total number of writebacks. 3) The proposed writeback-aware replacement policy has been able to reduce the writebacks originated from the *neutral* sets by 28.6%, on average (i.e., from 56.2% to 40.1% of the *Baseline* total writebacks). It is worth noting that for some of the benchmarks, the proposed replacement policy has also been able to reduce the number of writebacks originated from the *non-writer* sets. Figure 6 compares the normalized writebacks reduction of the evaluated schemes. Compared with *Baseline double-way* and *WADE*, WALL reduces the number of writebacks by 23.3% and 16.4%, on average, respectively. Based on the results, duplicating the sets associativity is not very helpful in reducing the number of LLC writebacks; our baseline implementation is 32-way and increasing the associativity beyond that does not cause a significant improvement.

C. *LLC Miss rate*

Figure 7 shows the normalized MPKI (Misses Per Kilo Instructions) of WALL for different types of LLC sets. The accesses that result in a hit in a *writer* set's partner are considered for the original *writer* set.

Results reveal that WALL reduces the MPKI by 2.4%, on average, compared to the *Baseline*. This reduction is mainly because WALL stores the frequently reused dirty blocks of *writer* sets in the *non-writer* partners. More specifically, the MPKI of *writer* sets is reduced by 27.8%, on average (i.e., from 30.7% to 22.1% of the *Baseline* total MPKI). On the other hand, the MPKI is increased from 12.0% to 16.2% for the *non-writer* sets and from 57.3% to 59.1% of the *Baseline* total MPKI for the *neutral* sets, on average, respectively. For some cases, the writeback-aware replacement policy has incurred miss penalties by evicting the clean lines that are later reused more frequently than the saved dirty blocks (i.e., those that have been given a second chance). For other cases, the dirty lines kept in LLC by the writeback-aware replacement policy are re-referenced more than the lines evicted by it instead. It should be noted that the penalty is small because we give only a second chance to the dirty victims of the sets. Figure 8 shows the normalized MPKI of the evaluated schemes. Compared with *Baseline double-way,* WALL increases the MPKI by 1.0%, on average. But, WALL reduces the MPKI by 0.3%, on average, compared to *WADE* because *WADE* does not distinguish different set types.

D. *Energy Comparison*

Figure 9 shows the normalized energy of the PCM main memory for the evaluated schemes. WALL can save main memory's energy by 19.2%, on average, compared to the *Baseline*. Compared with *Baseline double-way* and *WADE*, WALL reduces the energy consumption of the PCM main memory by 16.5% and 11.3% on average, respectively. The energy consumption of writing to PCM is much higher than that of reads. Hence, the energy saving is mainly due to the reduction in the number of write requests issued to PCM by WALL. As discussed in Section III, unlike *WADE*, which uses complex prediction schemes, the energy consumption of the logic components added to LLC by WALL is negligible.

E. *Performance Comparison*

Figure 10 compares the normalized system IPC of the evaluated schemes. The overhead of the second searches in partners of the *writer* sets is considered in our experiments.

WALL improves performance by 6.7% on average, compared to the *Baseline* scheme. Compared with *Baseline double-way* and *WADE*, WALL improves system performance by 4.9% and 3.2% on average, respectively. The latency of writing to PCM is much higher than that of reads. Hence, in addition to the reduced MPKI, the reduced average access latency of the PCM main memory is the other reason for the performance improvement. The results indicate that WALL can reduce the PCM average access latency by 12.6%, on average, compared to the *Baseline*. The reduction is due to reducing the PCM write traffic and thus the queuing delay of the PCM read requests.

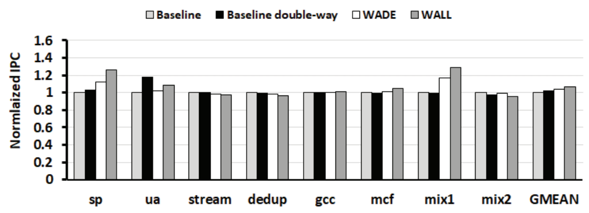

Figure 10. Normalized IPC

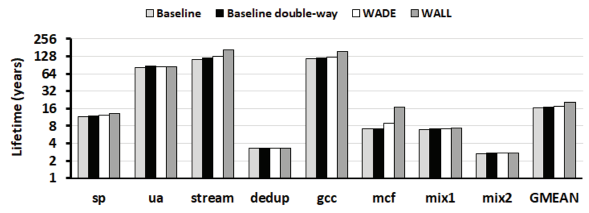

Figure 11. Lifetime enhancement (years, log scale).

F. PCM Lifetime Enhancement

Figure 11 compares the PCM lifetime enhancement of the evaluated schemes. The lifetime is calculated based on the analytical model explained in Section II.A. It should be noted that PCM lifetime is inversely proportional to the writes per cycle or write rate (GBps). WALL can enhance PCM lifetime by 1.25×, on average, compared to the *Baseline*. Compared with *Baseline double-way* and *WADE*, WALL enhances the PCM lifetime by 1.21× and 1.17× on average, respectively. Alleviating the write traffic sent out to the PCM main memory by WALL is the reason for the lifetime enhancement.

V. RELATED WORK

Many recent studies have focused on mitigating the overheads of write operations in PCM-based main memories by prioritizing reads over writes [15], parallelizing read/write accesses with an ongoing write [3, 23], merging multiple writes into one [19], etc. In this work, we focus on reducing the number of writes issued to PCM. Some studies have proposed techniques for reducing the number of LLC writebacks to the non-volatile component of a hybrid main memory consisting of NVM and DRAM [6, 21]. However, those techniques are only applicable for the hybrid main memories.

The set balancing cache (i.e., *SBC*) proposed in [17] tries to balance the pressure on cache sets by associating sets with maximum saturation counters with sets with small saturation counters. Though SBC reduces cache miss rate, it is not always able to reduce the number of writebacks. Our experiments show that, not every set with a maximum saturation counter has large number of writebacks (e.g., a set may evict clean lines mostly). On the other hand, there are some sets with saturation counters smaller than the maximum value, which write back more frequently than those with maximum saturation counter values. However, SBC shares resources between two sets only when a set's saturation counter reaches its maximum.

VI. CONCLUSIONS

In this paper, a novel writeback-aware LLC management scheme called WALL is proposed to reduce the number of LLC writebacks to a PCM based main memory. We first investigate the non-uniformity of the LLC sets writebacks, and based on that propose a writeback-aware set-balancing mechanism. We also propose a simple but effective writeback-aware replacement policy to keep the frequently reused dirty lines of the sets in the cache. Our evaluation results show that WALL can achieve a significant reduction in the total number of writebacks; thereby improving the system performance, energy efficiency and PCM lifetime.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under grant CCF-1513899. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] "CACTI 6.5," http: //hpl .hp.com: research /cacti/.
[2] "The NAS parallel benchmarks," http: // www. nas.nasa.gov/publications/npb.html.
[3] M. Arjomand *et al.*, "Boosting Access Parallelism to PCM-Based Main Memory," in ISCA, 2016.
[4] M. V. Beigi *et al.*, "TAPAS: Temperature-aware Adaptive Placement for 3D Stacked Hybrid Caches," in MEMSYS, 2016, pp. 415-426.
[5] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in PACT, 2009, pp. 72-81
[6] D. Chen *et al.*, "MALRU: Miss-penalty aware LRU-based cache replacement for hybrid memory systems," in DATE, 2017.
[7] Y. Choi *et al.*, "A 20nm 1.8v 8gb pram with 40mb/s program bandwidth," in ISSCC, 2012, pp. 46 - 48.
[8] X. Dong *et al.*, "Nvsim: A circuitlevel performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD,* vol. 31, no. 7, pp. 994 - 1007, 2012.
[9] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in ISCA, 2009, pp. 2-13.
[10] K. Lim *et al.*, "Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments," in ISCA, 2008, pp. 315-326.
[11] M. Poremba *et al.*, "NVMain 2.0: Architectural Simulator to Model (Non-)Volatile Memory Systems," *CAL*, no. 99, pp. 1, 2015.
[12] B. Pourshirazi *et al.*, "Refree: A Refresh-Free Hybrid DRAM/PCM Main Memory System," in IPDPS, 2016.
[13] B. Pourshirazi *et al.*, "NEMO: An Energy-Efficient Hybrid Main Memory System for Mobile Devices " in MEMSYS, 2017.
[14] M. K. Qureshi *et al.*, "Scalable high performance main memory system using phase-change memory technology," in ISCA, 2009, pp. 24-33.
[15] M. K. Qureshi *et al.*, "Improving read performance of Phase Change Memories via Write Cancellation and Write Pausing," in HPCA, 2010.
[16] S. Raoux *et al.*, "Phase-change random access memory: A scalable technolog," *IBM Journal of Research and Development* vol. 52, no. 4.5, pp. 465 - 479, 2008.
[17] D. Rolán *et al.*, "Adaptive line placement with the set balancing cache," in MICRO, 2009, pp. 529-540.
[18] Z. Wang *et al.*, "WADE: Writeback-aware dynamic cache management for NVM-based main memory system," *TACO,* vol. 10, no. 4, 2013.
[19] F. Xia *et al.*, "DWC: dynamic write consolidation for phase change memory systems," in ICS, 2014, pp. 211 - 220.
[20] C. Xu *et al.*, "Overcoming the Challenges of Crossbar Resistive Memory Architectures," in HPCA, 2015, pp. 476-488.
[21] D. Zhang *et al.*, "Write-back aware shared last-level cache management for hybrid main memory," in DAC, 2016.
[22] M. Zhang *et al.*, "Balancing Performance and Lifetime of MLC PCM by Using a Region Retention Monitor," in HPCA, 2017, pp. 385-396.
[23] P. Zhou *et al.*, "Throughput Enhancement for Phase Change Memories," *TC,* vol. 63, no. 8, pp. 2080 - 2093, 2014.