Write-back Aware Shared Last-level Cache Management for Hybrid Main Memory

Deshan Zhang Lei Ju* Mengying Zhao Xiang Gao Zhiping Jia
School of Computer Science and Technology
Shandong University, China
E-mail: {julei, jzp}@sdu.edu.cn

ABSTRACT

Hybrid main memory with both DRAM and emerging non-volatile memory (NVM) becomes a promising solution for high performance and energy-efficient embedded systems. Cache plays an important role and highly affects the number of write backs to NVM and DRAM blocks. However, existing cache policies fail to fully address the significant asymmetry between NVM operations (especially writes) and DRAM operations, leading to non-optimal system designs. We propose a write-back aware last-level cache management scheme for the hybrid main memory, which improves the cache hit ratio of NVM memory blocks and minimizes write-backs to NVM. Experimental results show that our proposed framework leads to better performance and energy saving compared with the state-of-the-art cache management scheme for hybrid main memory architecture.

1. INTRODUCTION

The SRAM/DRAM based memory has become energy and scalability bottlenecks of contemporary embedded systems with increasingly complex functionalities. Emerging non-volatile memory (NVM) technologies, including the spin-torque transfer magnetic RAM (STT-MRAM), resistive RAM (ReRAM), and phase change memory (PCM), are considered as attractive alternatives in the next generation memory hierarchy. Compared with traditional memory devices, NVM is capable to provide higher density, lower leakage power, as well as nonvolatility. However, current NVM devices typically suffer from higher access latency and dynamic power consumption, especially for the write operations due to their inherent nature.

In order to fully exploit the superiority of various technologies, hybrid main memory with both DRAM and NVM has been proposed as a promising solution for high performance and energy-efficient computer systems. In general, two architectures have been studied in the literature: (i) using a small and hardware-controlled DRAM as the buffer of the NVM-only main memory space (e.g., [14]); or (ii) having NVM and DRAM at the same level which constitute the overall main memory address space [15, 21]. In this work, we focus on the latter approach and refer it as the hybrid main memory architecture. The ultimate goal of hybrid main memory is to benefit from both DRAM's low write latency and NVM's high density as well as low static power consumption, in order to leverage the overall system performance and/or energy efficiency. To achieve that, most existing studies on hybrid main memory focus on dynamically migrating the physical pages between DRAM and NVM spaces (while keeping their virtual addresses unchanged) via various memory management schemes [4].

However, in a modern processor with caches, when the cache is physically indexed and/or tagged, moving pages between distinct physical memory devices in a hybrid main memory requires substantially high overhead including both memory copy as well as cache update time. Practically, cache management scheme not only determines the total number of cache misses and main memory accesses, but also has large impact on the miss ratio of individual memory blocks. As a result, designing the last-level cache (LLC) management policy is a cost-effective way to control the number of read and write operations on memory blocks located at NVM or DRAM in a hybrid main memory system. On the other hand, the LRU replacement policy is unaware of the significant asymmetry between NVM operations (especially writes) and DRAM operations, leading to non-optimal system designs. Wei et al. [18] proposed the first LLC management scheme HAP for hybrid main memory based on partitioning the cache lines between NVM and DRAM blocks. Since the read operations stall the processor while write operations can be buffered, HAP uses the corresponding read latencies to represent the cost of each cache miss to DRAM or NVM address space. However, as we will show in this paper, given the significantly higher NVM write latency and energy consumption, the write cost of evicting a dirty NVM block from LLC should be taken consideration in the cache management design.

In this paper, we propose a write-back aware shared LLC replacement scheme WBAR for high performance and energy efficient hybrid main memory architecture. In general, we design WBAR as a variant of LRU, which is light-weighted and easy to implement. Instead of inserting or promoting the latest visited cache block to the MRU (top priority) position as in LRU, WBAR places a cache block at distinct positions in a cache set according to the potential cost if the cache block is evicted from the cache before it gets reused. Experimental evaluation shows that the proposed scheme outperforms LRU and HAP [18] in terms of both performance and energy efficiency. Our contributions can be summarized as follows.

- We perform a set of empirical studies which shows even though NVM write operations (due to eviction of dirty NVM blocks from LLC) are buffered, they have significant impact on the overall performance and energy consumption in the hybrid memory system. We also show that number of hits on dirty memory blocks dominates the overall LLC demand hits for a large number of benchmarks. As a result, we show dirty block management is crucial for hybrid main memory system design.
- The proposed WBAR LLC management scheme decides the position of the latest visited block according to its state (clean or dirty), memory block type (NVM or DRAM), as well as the recent-history cache miss information (i.e., which type of the

memory devices incurs more cache misses in current program execution phase), in order to minimize the overall cost of the off-chip main memory accesses.

- The implementation of WBAR requires a (log_2A+1)-bit counter associated with each cache set to record the history miss information, and a 1-bit NVM/DRAM indicator for each cache block, where A represents the cache associativity. As a result, for a 8M-B LLC with 64B block size and 16-way associativity, the space overhead is only 0.26%. The update of the history counter at each LLC miss can be performed parallel with the cache content update, so that the performance overhead is negligible.

- We evaluate the proposed scheme with SPEC CPU 2006 benchmarks [2] running on an integrated simulation environment of gem5 [8] and NVMain [11]. Experimental results show that compared with LRU and HAP [18], WBAR improves the performance by 16.46% and 9.99%, and achieves 16.39% and 12.55% memory energy reduction, respectively.

2. RELATED WORK

The adoption of NVM as main memory encounters challenges of long write latency, high write energy, and/or limited write endurance. Comprehensive research work has been conducted to mitigate the write overhead for NVM-based main memory architecture. Lee et al. [1] use small row buffers to mitigate higher PCM array-write energy and utilize partial writes technique to enhance memory endurance. In order to reduce the impact of writes over read operations, Qureshi et al. [12] propose write cancellation and write pausing to delay the extremely slow write operations when read operations are waiting for service. Given the speed of DRAM and the higher density of NVM, hybrid main memory architecture has been proposed. Qureshi et al. [14] uses PCM as main memory and DRAM as the cache to filter access to PCM. The DRAM cache is hardware-controlled and transparent to the operating system. Another approach for hybrid main memory architecture is to combine the DRAM and NVM devices at the same level to constitute the entire memory address space [15, 21]. Lee et al. [4] propose a page-placement policy CLOCK-DWF to place frequently written data into DRAM, leaving rarely written ones into PCM, in order to reduce the number of write operations on PCM.

Cache plays a vital role in hiding the speed gap between processor and main memory. Effective use of the limited cache space, especially the shared last-level cache (LLC) in a multi-core system, is critical for high performance system design. Conventional cache replacement and partition policies aim to improve the overall cache hit rate, which implicitly treat all misses equally and distinguish *no* read or write operations. Recently, cost-sensitive replacement policies for LLC targeting DRAM based main memory system have been investigated. [6] divides the misses into *isolated misses* and *parallel misses*. Based on the non-uniformity in the performance impact of cache misses, it proposes a cost-aware cache replacement policy to reduce the number of performance-critical isolated misses. Besides, with the observation that cache read requests are frequently on the critical path and stall the processor whereas buffered write operations in the memory controller request queue are not, [17] proposes cache management mechanisms to increase the probability of cache hits for performance-critical reads, by allocating more cache resources to lines that are reused by reads than to lines that are reused only by writes.

Most existing design and optimization techniques for hybrid main memory with NVM and DRAM rely on memory page management schemes to mitigate the NVM write overhead. However, in a memory hierarchy with physically indexed and/or tagged on-chip caches, page migrations between different memory devices require additional overhead to locate and update the states of corresponding cache blocks. On the other hand, design of the LLC management policy is a cost-effective way to control the actual number of read and write operations on distinct memory address spaces. Recently, there have been LLC cache management policies proposed to reduce write-back requests for a NVM-only main memory. Zhou et al. [7] proposed write-aware policy (WCP) to partition the shared LLC among multiple cores by taking into account the write-back penalty. The proposed policies in [16] and [20] set different priority for each cache block based on block's state (e.g., clean or dirty). They prioritize clean blocks as a victim when replacement happens. However, these polices can't be directly applied to the hybrid memory architecture in which the memory address space contains both DRAM and NVM.

Wei et al. [18] take the first step to exploit the LLC partitioning and replacement policy in the context of hybrid main memory. The proposed HAP scheme considers the miss cost disparities between DRAM and NVM and logically divides the LLC into two partitions. Similarly as the strategy in [17] for DRAM-based memory, HAP considers read operations on the critical path of performance since they stall the processor. Furthermore, with the write-allocate policy, either a CPU read miss or a write miss triggers a memory read. As a result, HAP represents the cost of NVM operations with the NVM read latency. However, as we will show in this paper, given the significant higher write latency and energy consumption of current NVM technology compared with DRAM writes, NVM write operations need to be mitigated in a hybrid memory architecture even with the memory request queue. Therefore, eviction of dirty NVM blocks from the LLC needs to be managed subtly.

3. MOTIVATION

Conventional cache management policies like LRU target at minimizing overall cache miss ratio. They implicitly assume that different miss operations to the next level of memory hierarchy lead to identical cost. However, a cache read miss may stall the processor, hence has larger impact on the performance compared with the non-blocking write operations buffered in the memory request queue. Recent work shows that an improved LLC replacement policy results in a 6% system speedup by taking consideration of the read-write disparity in a DRAM based memory system [17]. On the other hand, in a hybrid main memory with both DRAM and NVM devices, significant disparity can be observed between the read and write operations on DRAM or NVM cells. For example, PCM read and write latencies (energy consumption) are approximately 4.4X (2.1X) and 12.0X (43.1X) greater than those of DRAM [1]. As a result, each LLC miss may have distinct cost depending on the resultant operations on the DRAM/NVM devices, which brings a great challenge for design and optimization of the LLC management scheme for hybrid main memory.

The hybrid-memory-aware LLC management scheme (a.k.a HAP) proposed in [18] uses the NVM read latency as the dominant cost for NVM related operations since read operations may stall the processor execution. While the claim is generally true for DRAM based memory system ([17]), we observe that the long NVM write latency and high write energy have significant impact on the overall system performance and energy consumption. In the case of the long write latency, it is more likely that the memory request queue becomes fully occupied, which blocks the subsequent read and write operations. Figure 1 shows the performance for a set of workloads consisting of SPEC CPU 2006 benchmarks [2] on a hybrid main memory with DRAM and PCM. We assume a 4-core system with an 8MB shared LLC under LRU, the DRAM and PCM controller each has a request queue with 64 slots, and the read/write latencies for DRAM and PCM are set to 1:1:4.4:12 as in [1]. Details of the workloads and other experimental setup can be found in Section 5. In order to evaluate the impact of NVM write latency and write energy on the overall system performance and energy, we set the PCM array (cell) write latency and energy to be 0 in the system simulation. The normalized performance and energy improvements of the 0-PCM-write-latency system ("0-write-latency") and the 0-PCM-write-energy system ("0-write-energy") w.r.t. the normalized values of the original system are illustrated in Figure 1. As shown in the results, PCM array writes contribute 21% of the total execution time in average, hence become one of the key considerations in the performance optimization. Similarly, PCM array write energy contributes 34% of the total energy in average.

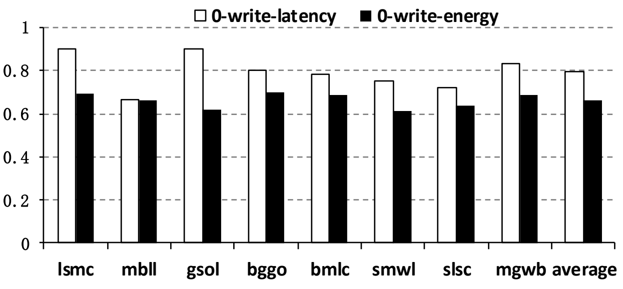

Figure 1: **The performance and energy ratio of "0-write-latency" and "0-write energy" w.r.t. the normal system.**

Given the above-mentioned observation, we focus on reducing the number of NVM array writes in order to achieve overall performance and energy improvement in this work. A NVM array write is typically caused by writing-back *dirty* memory row buffer content upon a row buffer miss. Existing study shows that NVM benefits from a small row buffer size for reduced NVM array read and write energy. However, when the chip row buffer size is 64B, the row buffer miss ratio in an 8-core system is over 65% [5]. With the commonly applied write-back cache policy, it is crucial to reduce the write-backs of dirty cache blocks to the NVM row buffer in order to reduce the NVM array writes.

4. CACHE REPLACEMENT POLICY

In this section, we first investigate the typical LLC behavior, which leads to our priority assignments to various types of LLC blocks. We then present the design and implementation of our light-weighted write-back aware LLC replacement policy (WBAR), followed by an illustrative example to demonstrate the proposed scheme.

4.1 Cache Behavior Analysis

With the increasing gap between the CPU and main memory speed, cache replacement policy becomes a critical design consideration for high performance computer systems. The widely-used Least-Recently Used (LRU) replacement policy arranges blocks using a *recency stack*. In particular, the least recently used cache block occupies the lowest priority position (LRU), whereas the newly visited block is inserted in the highest priority position (MRU). However, existing study shows that more than 60% of the blocks in LLC lead to *no* cache hits with LRU [13]. Furthermore, since LRU is unaware of the asymmetric performance and energy cost between DRAM and NVM accesses, it may not lead to an optimal system design for hybrid main memory architecture. In this work, we consider the following two design goals for LLC replacement policy with hybrid main memory architecture:

- The LLC replacement policy should reduce the overall cache misses so that the total main memory accesses are minimized.
- The LLC replacement policy should be aware of the cost of various memory operations of a hybrid main memory.

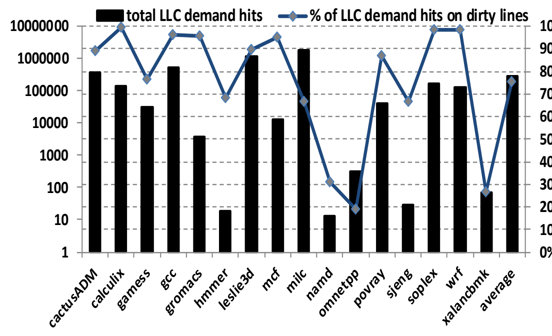

Figure 2: **the demand hit ratio of dirty cache blocks**

In a multi-level cache system, an access to the LLC can be classified as a demand access (i.e., a CPU read/write request results in misses in all upper-level caches), or a write-back access (the immediate upper-level cache evicts a dirty block which needs to be inserted into LLC). An LLC demand miss leads to a main memory read which stalls the CPU execution; whereas a write-back miss introduces a new dirty block in the LLC and triggers *no* memory reads. In general, demand misses have higher impact on the system performance due to the CPU-stalling memory read. In Figure 2, we show the demand access behavior on LLC for a set of SPEC CPU 2006 benchmarks under LRU. The detailed gem5 simulation environment is presented in Section 5. Results show that on average, 75.3% percentage of all LLC demand hits across various benchmarks happen on the dirty cache blocks. For example, *calculix* has over 130000 hits within the simulation period, including 99.6% dirty block hits. The result suggests that dirty cache blocks have a much higher chance to be referenced.

In addition to the memory read operations resulting from LLC demand misses, a demand or write-back miss in a fully-occupied LLC cache set triggers a cache eviction, which may lead to a memory write request if the evicted block is dirty. As we have shown in Section 3, due to the long write latency and high write energy consumption of NVM in a hybrid main memory, memory write operations have significant impact on the overall system performance and energy efficiency.

In this work, we categorize cache blocks in LLC into 4 types: Dirty-NVM (DN), Dirty-DRAM (DD), Clean-NVM (CN), Clean-DRAM (CD), indicating dirty/clean cache blocks from NVM or DRAM, respectively. Given the above-mentioned observations, we set the LLC priorities for different types of cache blocks as follows:

$$DNP > DDP > CNP > CDP$$

, where DNP, DDP, CNP, and CDP are the LLC priorities for DN, DD, CN, and CD blocks, respectively. In general, dirty LLC blocks have higher chances to be re-referenced, and higher cost to be evicted. As a result, we would like to keep a dirty NVM in LLC for longer time compared to other blocks. And given the NVM read latency is longer than DRAM read, if a NVM clean block is evicted from LLC and gets re-referenced again later, it leads to a higher performance cost than that of a DRAM clean block.

4.2 WBAR scheme

The standard LRU replacement policy can be divided into three sub-policies: insertion, promotion, and eviction [19]. In order to reflect the different priority levels for various block types as we have defined, a variant of LRU can be used by modifying the insertion and promotion sub-policies of LRU. In particular, an incoming new cache block or a hitting cache block should be inserted or promoted according to its block type and associated priority, instead of being placed at the MRU position as in LRU.

Given the fact that an application may have distinct memory behaviors at different execution phases (e.g., entering a different loop or procedure), the priority of each cache block type (as we have defined in Section 4.1) should be modified dynamically to cope with such changes. Many LLC policies use Set Dueling/Sampling to achieve a dynamic management mechanism (e.g., [3, 17, 18, 19]). However, as suggested in [3], set dueling requires many sampling sets in order to obtain an optimal design choice when the design space is large (e.g., the possible insertion location in our case). Furthermore, it needs relatively complex logic circuit to manage the sampling interval, sampling mechanism, as well as the decision of best policy and application to the rest cache sets.

We use the following notions and assumptions in description of the proposed cache management scheme. We equip each cache set with a n-bit saturating counter $counter$, where $n = (log_2 A + 1)$ and A is the associativity of LLC. The initial value of the counter is set to be 2^{n-1}. We also assume the associativity of the shared LLC is at least 8. we assume the highest priority position (MRU) is at *A-1*, and an LLC eviction always chooses the LRU block at position 0. Our WBAR scheme is illustrated in Algorithm 1.

If a request r leads to a write-back hit (the write-back from immediate upper-level cache hits in LLC) for cache block C in LLC, C is set to be a dirty block, and the data is updated accordingly (line 2-3). For the promotion sub-policy, if the hitting cache block C is a NVM block, it is promoted to the MRU position in the cache set (line 5-6). Otherwise if C is a DRAM block, it is promoted by (at most) DDP over its current position $C.pos$. If r results in a cache miss, an incoming cache block needs to be inserted (according to write-back and write-allocation mechanism), and we first evict the cache block at LRU position (line 12). If r is a write-back miss, a dirty block from upper-level cache (i.e., $r.block$) is to be inserted. The insertion location depends on whether it is a NVM or DRAM block (line 14-20). Otherwise, r is a demand miss and $r.block$ is a clean memory block to be read from main memory. Similarly, its insertion location is calculated according to its memory device type (line 22-28).

Note that the saturating counter gets decreased by 1 for a demand miss on NVM (line 23), and increased by 1 for a demand miss on DRAM (line 26). Intuitively, a larger counter value indicates more DRAM demand misses in the recent history, so that the priorities of DRAM blocks during insertion and promotion should be raised to potentially improve the LLC hit ratio of DRAM blocks. In general, the saturating counter based WBAR scheme is light-weighted and easy-to-implement, which leads to less than 0.3% space overhead for 16-way associative LLC as discussed in Section 5.4. The effectiveness of the proposed cache management scheme will be illustrated in Section 5.
Figure3 illustrates an example of WBAR for an 8-way LLC cache, where we denote each cache block type with different symbols. The saturating counter has *4* bits, and its current value is *1001B* before processing any requests in the example. When DRAM block *9* is evicted from upper lever cache and triggers a write-back miss in LLC, it is inserted at the DDP position (i.e., *4* according to line 18 of Algorithm 1). The saturating counter remains unchanged with such a write-back miss. However, a read (demand) miss of DRAM block *3* increases the saturating counter by 1. The saturating counter is later deducted by 1 due to the read miss of NVM block *C*. As for read request for DRAM block *5*, it is a cache hit and the corresponding cache block is promoted by *4* positions determined by DDP (line 8-9 of Algorithm 1). On a read request hit for NVM block *E*, the cache block is promoted to the MRU position (line 6 of Algorithm 1). Whenever a cache block needs to be evicted, the cache block at the LRU position is always selected.

Algorithm 1: WBAR(r) — r denotes a request to LLC.

```
1:  if $r$ hits in cache block $C$ then
2:      if isWriteback($r$) then
3:          $C.dirty$ = 1; updateContent($C$);
4:      end if
5:      if isNVM($C$) then
6:          promote($C$, $A - 1$);/*promote cache block $C$ to MRU position*/
7:      else
8:          $DDP = counter/2$; $pos = C.pos + DDP$;
9:          promote($C, pos > MRU?MRU : pos$);
10:     end if
11: else
12:     evictCache(0); /*evict the cache block at the LRU position in same set*/
13:     if isWriteback($r$) then
14:         if isNVM($r.block$) then
15:             $DNP = A - 1 - counter/8$;
16:             insert($r.block$,$DNP$);
17:         else
18:             $DDP = counter/2$;
19:             insert($r.block$,$DDP$);
20:         end if
21:     else
22:         if isNVM($r.block$) then
23:             $counter$–;
24:             $CNP = counter/4$; insert($r.block$,$CNP$);
25:         else
26:             $counter$++;
27:             $CDP = counter/8$; insert($r.block$,$CDP$);
28:         end if
29:     end if
30: end if
```

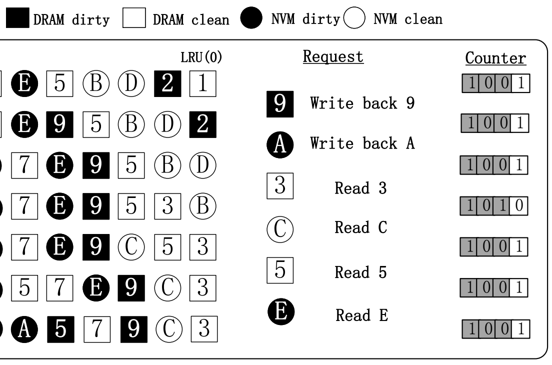

Figure 3: Illustrative examples of WBAR.

5. EVALUATION

5.1 System Configuration

We use gem5 [8] as our simulation platform with the integration of NVMain [11], which is a cycle accurate memory simulator for both DRAM and non-volatile memories (NVM). Table 1 shows the detailed system configuration. For the 3-level cache hierarchy, we use the write-back and write-allocation policy, and the classic non-inclusive cache model in gem5. We use NVMain to construct two main memory modules, 128MB DRAM and 1GB NVM. The NVM module is based on the PCM prototype. Each memory module has a request queue which supports a maximum of 64 outstanding requests. Table 2 shows the energy and latency parameters of PCM and DRAM as from [1]. The array access energy and buffer access energy represent the energy consumed for data array access and row buffer access per bit, respectively.

Table 1: System configurations.

Processor	4-cores, Out-of-Order, 2GHz
L1(per core)	64KB I-cache, 64KB D-cache, 2-cycle latency, 4-way, 64B/line
L2(per core)	256KB, 8-way, 64B/line, 10-cycle latency
LLC(shared)	8MB, 16-way, 64B/line, 30-cycle latency
DRAM	128MB, open-page, FR-FCFS row-buffer=1KB
NVM	1GB, open-page, FR-FCFS row-buffer=64B [1, 9]

Table 2: Timing and Energy parameters of NVM and DRAM [1]

		NVM	DRAM
Energy (pJ/bit)	Array read	2.47	1.17
	Array write	16.82	0.39
	Buffer read	0.93	0.93
	Buffer write	1.02	1.02
Latency(ns)	tRCD	55	12.5
	tCL	12.5	12.5
	tWTR	7.5	7.5
	tRP	150	12.5

According to [10] and [4], for the initial memory mapping, we place read-only data (the code) and infrequently updated data (the static initialized data and static uninitialized data in BSS) into NVM memory address space. As for heap data and stack data, considering the small DRAM space, they are randomly placed into either DRAM or NVM memory. For each cache set, we initialize the 5-bit saturating counter to be *10000B*.

5.2 Workload

We select memory-intensive benchmarks from SPEC CPU2006 [2] as our workloads. Table 3 shows the characteristics of all workloads. We do not consider benchmarks with L2-MPKI less than one for our studies as they do not exert significant pressure on the last-level cache. For each workload, we first run all benchmarks for 20 million instructions for cache warmup and then the following 500 million instructions for the statistics.

Table 3: Workloads characteristics.

Workloads	Description	L2-MPKI
lsmc	leslie3d, soplex, milc, cactusADM	9.89
mbll	mcf, bzip2, libquantum, lbm	5.76
gsol	gromacs, soplex, omnetpp, leslie3d	1.02
bggo	bzip2, gromacs, gobmk, omentpp	1.48
bmlc	bzip2, milc, libquantum, cactusADM	14.76
smwl	soplex, mcf, wrf, lbm	2.57
slsc	sjeng, lbm, soplex, cactusADM	7.00
mgwb	milc, gromacs, wrf, bwaves	7.51

5.3 Evaluation Results

For all comparisons, we use the results with the conventional LRU replacement as the baseline, and show the normalized results for HAP [18] and the proposed WBAR.

5.3.1 Performance Evaluation

Figure 4 compares the performance, instructions per cycle (IPC), for all workloads with different mechanisms. On average, WBAR improves performance by 16.46% compared with LRU, and 9.99% with respect to HAP. The reason lies in that, compared with LRU, WBAR differentiate DRAM and NVM in insertion and promotion priority; and compared with HAP, WBAR further explored the disparity between dirty and clean cache blocks from both DRAM or NVM.

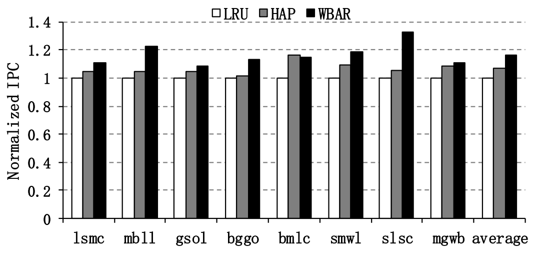

Figure 4: Normalized IPC speedup.

Figure 5 summarizes the normalized write-back operations to DRAM and NVM for LRU, HAP and WBAR. The combination of Figure 4 and 5 shows that workloads that have large percentage of reduced NVM write-backs relate significant performance improvements. For example, in *slsc*, WBAR reduces NVM write-back by 18.91% when compared with LRU and improves the system IPC by 32.94%. Even *gsol* relates to more DRAM write-backs than LRU, it still achieves improved IPC due to reduced NVM write-backs. This observation confirms the large impact of NVM write on system performance. WBAR can effectively reduce the write-back requests because it sets relatively higher priority for dirty data than clean data, especially retaining NVM dirty data in cache for a much longer time. On average, WBAR reduces the NVM write-backs by 11.47% when compared with HAP and 17.81% when compared with LRU. The reduced number of NVM write-backs benefit the system performance as well as energy.

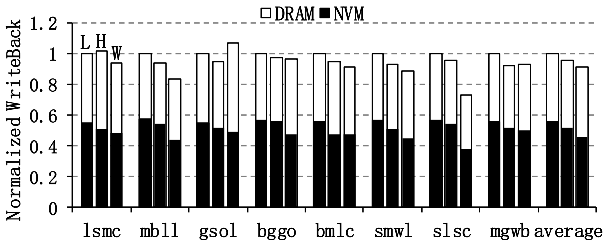

Figure 5: Normalized Write Back.L=LRU, H=HAP, W=WBAR

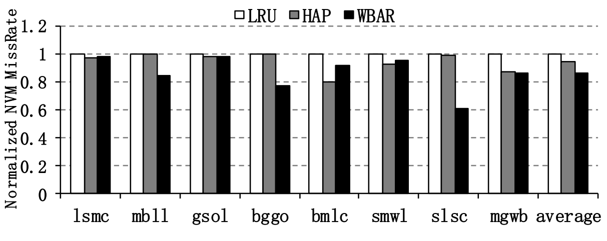

Figure 6: Normalized NVM missrate.

In order to take a deep look at the insight, we summarize the normalized NVM miss rate in Figure 6 and DRAM miss rate in Figure 7. The observations are as follows. Firstly, the proposed WBAR, on average, achieves the lowest miss rate in both NVM and DRAM. Compared with LRU, WBAR reduces the NVM and DRAM miss rate by 13.6% and 11.27%, respectively. This indicates that, in WBAR, rasing the priority of dirty NVM in LLC cache does not degrade the cache behavior in general. Secondly, since both HAP and WBAR assign NVM blocks higher priorities, they may potentially lead to less space for DRAM blocks in LLC. That is why HAP and WBAR have higher DRAM miss rates for some benchmarks as shown in Figure 7. But WBAR still outperforms HAP in DRAM miss rate on average. The reason lies in that HAP equally regards

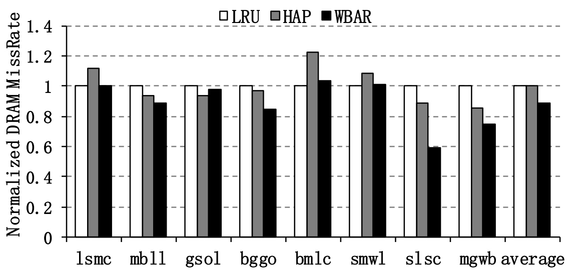

Figure 7: Normalized DRAM miss rate.

dirty and clean DRAM blocks while WBAR pays more attention on dirty DRAM blocks. On the basis of the observation that dirty blocks have higher hit ratio than clean ones, WBAR wins by maintaining more dirty DRAM blocks in LLC. Thirdly, for special case such as *bmlc*, even it has higher NVM miss rate in WBAR than HAP, the less write-backs and lower DRAM miss rate in WBAR lead to comparable system performance with HAP.

5.3.2 Energy Evaluation

Figure 8 shows the energy consumption of the different approaches compared to the baseline LRU policy for all workloads, respectively. It also shows the breakdown of read/write energy (read-/write to row buffer), activation/precharge energy (activation/write to array) and background energy (including DRAM refresh energy). For NVM main memory, the activation/precharge energy consumption dominates the NVM main memory energy consumption and is higher than that for DRAM, and thus activation/precharge energy accounts for a large proportion of total energy. In WBAR, the obvious reduction in the NVM write back requests significantly reduces the activation/precharge energy consumption in NVM main memory. Besides, WBAR reduces the total execution cycles of workloads, leading to the reduction in background energy, especially the DRAM refresh energy. As shown in Figure 8, compared with LRU and HAP, the proposed WBAR achieves an average memory subsystem energy reduction of 16.39% and 12.55%, respectively.

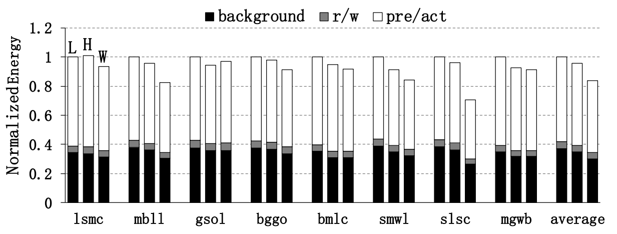

Figure 8: Normalized energy saving. L=LRU, H=HAP, W=WBAR

5.4 Overhead Analysis

For given cache configuration in our experimental setting as shown in Table 1, with cache line size of 64B and associativity of 16, we requires a 5-bit saturating counter for each cache set and 1-bit memory type indicator for each cache line. Therefore, the storage overhead is $5/(16 \times 64 \times 8) + 1/(64 \times 8)$, which is approximately 0.26% of the overhead LLC capacity.

The performance overhead for the counter update with each cache miss can be conducted parallel with the main memory access, therefore is negligible. The insertion or promotion of cache lines in the LLC requires to read the counter of the corresponding cache set, which can be implemented with a simple and fast circuit.

6. CONCLUSION

In this paper, we propose a light-weighted write-back aware LLC replacement scheme WBAR to improve the overall LLC hit ratio and mitigate the LLC miss cost for hybrid main memory architecture. WBAR dynamically assigns different priorities to cache blocks in LLC according to the potential cost if the cache block is evicted from the cache before it gets reused. Experiment results show that WBAR efficiently improve system performance as well as energy saving with a negligible storage and timing overhead.

7. ACKNOWLEDGMENTS

This research is sponsored by the Natural Science Foundation of China (NSFC) under Grant no. 61202015 and 61533011, Shandong Provincial Natural Science Foundation under Grant no. ZR2013FM028 and ZR2015FM001, the Fundamental Research Funds of Shandong University under no. 2015JC030.

8. REFERENCES

[1] B.C.Lee,et al. Architecting phase change memory as a scalable DRAM alternative. *ACM SIGARCH Computer Architecture News*, 37(3):2–13, 2009.
[2] J. L. Henning. SPEC CPU 2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.
[3] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *ACM SIGARCH Computer Architecture News*, volume 38, pages 60–71. ACM, 2010.
[4] S. Lee, H. Bahn, and S. H. Noh. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures. *IEEE Transactions on Computers*, 63(9):2187–2200, 2014.
[5] J. Meza, J. Li, and O. Mutlu. A case for small row buffers in non-volatile main memories. In *IEEEqqInternational Conference on Computer Design (ICCD)*, 2012.
[6] M.K.Qureshi,et al. A case for MLP-aware cache replacement. *ACM SIGARCH Computer Architecture News*, 34(2):167–178, 2006.
[7] M.Zhou,et al. Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):53, 2012.
[8] N.Binkert,et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.
[9] H. Park, S. Yoo, and S. Lee. Power management of hybrid dram/pram-based main memory. In *Design Automation Conference (DAC)*, pages 59–64, 2011.
[10] Y. Park, S. K. Park, and K. H. Park. Linux kernel support to exploit phase change memory. In *Linux Symposium*, volume 2010, pages 217–224, 2010.
[11] M. Poremba and Y. Xie. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 392–397. IEEE, 2012.
[12] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montaño. Improving read performance of phase change memories via write cancellation and write pausing. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–11. IEEE, 2010.
[13] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 381–391. ACM, 2007.
[14] 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*, 37(3):24–33, 2009.
[15] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page placement in hybrid memory systems. In *the International conference on Supercomputing*, 2011.
[16] R. Rodríguez-Rodríguez, F. Castro, D. Chaver, R. Gonzalez-Alberquilla, L. Piñuel, and F. Tirado. Write-aware replacement policies for pcm-based systems. *The Computer Journal*, 2014.
[17] S.Khan,et al. Improving cache performance by exploiting read-write disparity. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 452–463, 2014.
[18] W. Wei, D. Jiang, J. Xiong, and M. Chen. Hap: Hybrid-memory-aware partition in shared last-level cache. In *IEEE International Conference on Computer Design (ICCD)*, pages 28–35. IEEE, 2014.
[19] Y. Xie and G. H. Loh. Pipp: promotion/insertion pseudo-partitioning of multi-core shared caches. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 174–183. ACM, 2009.
[20] X.Zhang,et al. A read-write aware replacement policy for phase change memory. In *Advanced Parallel Processing Technologies*, pages 31–45. 2011.
[21] W. Zhang and T. Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 101–112. IEEE, 2009.