WADE: Writeback-Aware Dynamic Cache Management for NVM-Based Main Memory System

ZHE WANG, Texas A&M University
SHUCHANG SHAN, Chinese Institute of Computing Technology
TING CAO, Australian National University
JUNLI GU and YI XU, AMD Research
SHUAI MU, Tsinghua University
YUAN XIE, AMD Research/Pennsylvania State University
DANIEL A. JIMÉNEZ, Texas A&M University

Emerging Non-Volatile Memory (NVM) technologies are explored as potential alternatives to traditional SRAM/DRAM-based memory architecture in future microprocessor design. One of the major disadvantages for NVM is the latency and energy overhead associated with write operations. Mitigation techniques to minimize the write overhead for NVM-based main memory architecture have been studied extensively. However, most prior work focuses on optimization techniques for NVM-based main memory itself, with little attention paid to cache management policies for the Last-Level Cache (LLC).

In this article, we propose a **W**riteback-**A**ware **D**ynamic Cach**E** (WADE) management technique to help mitigate the write overhead in NVM-based memory.[1] The proposal is based on the observation that, when dirty cache blocks are evicted from the LLC and written into NVM-based memory (with PCM as an example), the long latency and high energy associated with write operations to NVM-based memory can cause system performance/power degradation. Thus, reducing the number of writeback requests from the LLC is critical.

The proposed WADE cache management technique tries to keep highly reused dirty cache blocks in the LLC. The technique predicts blocks that are frequently written back in the LLC. The LLC sets are dynamically partitioned into a frequent writeback list and a nonfrequent writeback list. It keeps a best size of each list in the LLC. Our evaluation shows that the technique can reduce the number of writeback requests by 16.5% for memory-intensive single-threaded benchmarks and 10.8% for multicore workloads. It yields a geometric mean speedup of 5.1% for single-thread applications and 7.6% for multicore workloads. Due to the reduced number of writeback requests to main memory, the technique reduces the energy consumption by 8.1% for single-thread applications and 7.6% for multicore workloads.

Categories and Subject Descriptors: B.3.2 [**Design Styles**]: Cache Memories

General Terms: Design, Performance

Additional Key Words and Phrases: Last-level cache, nonvolatile memory, replacement policy, cache segmentation

[1]WADE means "walk through water with difficulty because of the pressure of the water against the legs." We envision our WADE technique as relieving the pressure of write overheads for NVM-based memory architecture.

Part of this work was conducted while Zhe Wang was on an internship at AMD Research.
This work is supported in part by the National Science Foundation, under grants CCF-1332654, CCF-1332598, CCF-1332597, CCF-1218867, and CCF-1213052.
Author's address: Z. Wang, Computer Science and Engineering Department, Texas A&M University.

DOI: http://dx.doi.org/10.1145/2555289.2555307

ACM Reference Format:
Wang, Z., Shan, S., Cao, T., Gu, J., Xu, Y., Mu, S., Xie, Y., and Jiménez, D. A. 2013. WADE: Writeback-aware dynamic cache management for NVM-based main memory system. ACM Trans. Architec. Code Optim. 10, 4, Article 51 (December 2013), 21 pages.
DOI: http://dx.doi.org/10.1145/2555289.2555307

1. INTRODUCTION

Technology scaling of SRAM and DRAM is increasingly constrained by fundamental technology limits. Emerging memory technologies, such as Spin Torque Transfer RAM (STT-RAM), Phase-Change RAM (PCM), and Resistive RAM (RRAM), are being explored as potential alternatives to existing memories in future computing systems. Compared to the traditional SRAM/DRAM technology, these emerging memories have the common advantages of high density, low standby power, better scalability, and nonvolatility, and hence become very attractive as the alternatives for future memory hierarchy.

In order to use such emerging memories as main memory, several design issues must be solved. The most important is the performance and energy costs of writes. Since NVM has an inherently stable mechanism for data storage, it takes more time and energy to overwrite data. Write endurance is another challenge in PCM-based and RRAM-based main memory systems. For example, state-of-the-art process technology has demonstrated that the write endurance for PCM is around 10^8 to 10^9 writes. The problem is further aggravated by the fact that the service of memory write requests consumes power and delays the service of subsequent memory read requests, causing performance degradation [Stuecheli et al. 2010; Wang et al. 2012; Lee et al. 2010]. This write inefficiency problem is worse in NVM-based main memory. Thus, if we directly replace DRAM with NVM as main memory, the long latency and high energy consumption of writes could offset performance and power benefits and result in degradations. Much previous work has investigated techniques to mitigate the overheads of write operations, such as read-before-write, load balancing, novel write-buffer structure, or hybrid cache/memory architecture [Joo et al. 2010; Qureshi et al. 2009; Sun et al. 2009; Ramos et al. 2011], which will be discussed in Section 5. However, the majority of these techniques focus on optimization techniques for NVM-based main memory itself, paying little attention to cache management of the LLC.

This article focuses on evaluating the impact of LLC management policies on the memory performance/power and tailors the cache replacement policy to the underlying NVM-based main memory. Caches are used to mitigate the performance gap between the main memory and processor. A variety of dynamic cache management techniques [Jaleel et al. 2008; Qureshi et al. 2007; Jaleel et al. 2010; Qureshi et al. 2006; Khan et al. 2012; Xie and Loh 2009; Chang and Sohi 2007; Qureshi and Patt 2006] have been proposed to improve cache efficiency. Most of these techniques provide better performance by dynamically learning access patterns, selecting the management policy, or partitioning the cache capacity. Dirty cache blocks evicted from the LLC will be written into main memory, which can degrade performance and energy efficiency, especially in an NVM-based main memory system. Clean cache blocks will not affect the system performance and energy once they are evicted from the LLC. Most previous cache management policies are not aware of the disparity in miss penalty for dirty and clean cache blocks. Zhou et al. [2012] take the first step to exploit the LLC partitioning and replacement policy by considering the negative impact of writeback requests. They propose to partition the shared LLC among multicore by taking into account the writeback penalty. However, their technique can only be applied to multicore systems with the Least-Recently-Used (LRU) replacement policy.

This article introduces the Writeback-Aware Dynamic cachE (WADE) management technique in the context of NVM-based main memory. WADE improves system performance and energy efficiency by reducing the number of writeback requests to NVM-based main memory. The technique is decoupled from the LLC replacement policy. It tries to keep highly reused dirty data in the LLC. LLC blocks are classified into *frequent writeback blocks* and *nonfrequent writeback blocks*. The *frequent writeback blocks* are cache blocks in the LLC that are written back to main memory with high frequency within a certain access interval. The remaining cache blocks are classified as *nonfrequent writeback blocks*. The *nonfrequent writeback blocks* can be clean cache blocks or dirty cache blocks that are not written into main memory frequently. The cache set is dynamically partitioned into a *frequent writeback list* and a *nonfrequent writeback list*. WADE improves cache efficiency by trying to keep the best size of each list. The evaluation shows the technique can improve the system performance and reduce the energy consumption for both single-thread and multicore workloads. Our contributions can be summarized as follows:

—We investigate write latency and energy consumption in the context of PCM-based main memory, with analysis of the disparity in miss penalties for different types of data. We propose to differentiate the miss penalty for dirty cache blocks and clean cache blocks.
—We propose a frequent-write predictor that predicts frequent writeback cache blocks in the LLC. It takes into account both recency and frequency information and makes a prediction at both coarse granularity and fine granularity.
—We propose to partition LLC sets into a frequent writeback list and a nonfrequent writeback list. The technique is aware of the characteristics of each list and dynamically learns the optimal size of each list.
—We present an evaluation of our technique with SPEC CPU 2006 [Henning 2006] workloads simulated with the MARSSx86 [Patel et al. 2011] simulator combined with a modified DRAMSim2 [Rosenfeld et al. 2011] for simulating PCM-based main memory. Our evaluation shows that the technique yields a geometric mean speedup of 5.1% for memory-intensive single-thread applications and 7.6% for multicore workloads. The techniques reduce the energy consumption in main memory by 8.1% for single-thread applications and 7.6% for multicore workloads.

The rest of the article is organized as follows: Section 2 gives the background of emerging nonvolatile memories, the impact of write on the PCM-based memory system, and motivational results. Section 3 explains our writeback-aware dynamic cache management technique. Section 4 presents our experimental methodology and results. Section 5 discusses related work. Finally, Section 6 concludes the article.

2. BACKGROUND AND MOTIVATION

In this section, we first give a brief introduction on emerging nonvolatile memory and the impact of memory write overhead on both performance and power. We then discuss the motivation of our research with an example that demonstrates two interesting observations, which shows the potential of our proposed WADE management technique.

2.1. Emerging Nonvolatile Memory

Even though our proposed technique works for DRAM-based memory systems, it works much better for emerging NVM-based memory systems. Therefore, we first give a brief introduction on emerging nonvolatile memory technologies.

In recent years, significant efforts and resources have been put into the research and development of emerging memory technologies that combine attractive features such as scalability, fast read/write, negligible leakage, and nonvolatility. Multiple promising

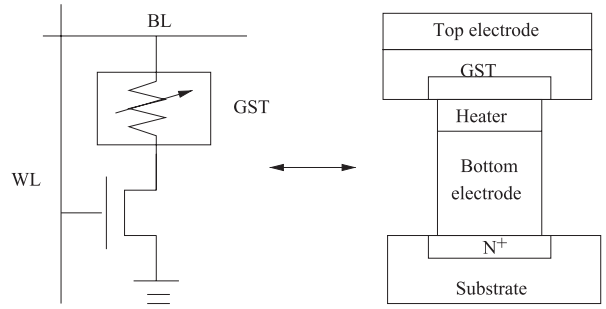

Fig. 1. An illustration of Phase-change RAM (PCM) cell. The GST has two phases: the amorphous phase with high resistance and the crystalline phase with low resistance.

candidates, such as PCM, STT-RAM, and RRAM, have gained substantial attention and are being actively pursued by the industry [Raoux et al. 2008]. Among various emerging memory technologies, PCM is one of the most promising candidates because semiconductor companies have made dramatic R&D progress in recent years. For example, Samsung demonstrated an 8Gbit PCM memory chip recently [Choi et al. 2012], and a CMOS-compatible embedded PCM (Hitachi and STMicro) [Hanzawa et al. 2007; Pellizzer et al. 2004] has been demonstrated, paving the way for integrating these NVMs into traditional memory hierarchies. In addition, emerging 3D integration technologies [Sun et al. 2009] enable cost-effective integration of these NVMs with CMOS logic circuits. Compared with DRAM, the PCM [Xie 2011] has high density, comparable read access time, and reasonable write endurance, which made it a promising alternative to existing main memories. Thus, many innovative memory architectures using PCM as main memory have emerged in the last several years [Qureshi et al. 2010; Xie 2011; Qureshi et al. 2009, 2009; Zhou et al. 2009; Lee et al. 2009].

In a PCM memory cell, the storage node is based on a chalcogenide alloy (typically GeSBTe (GST) material), as shown in Figure 1. The resistance differences between an amorphous (high-resistance) and crystalline (low-resistance) phase of chalcogenide-based material indicate the stored value as "1" and "0," respectively. Writing a bit to the PCM cell is done through *set* and *reset* operations: for set operations, the phase-change material is crystallized by applying an electrical pulse that heats a significant portion of the cell above its crystallization temperature. In reset operations, a larger electrical current is applied and then abruptly cut off to melt and then quench the material, leaving it in an amorphous state. Compared to charge-based SRAM/DRAM, PCM intrinsically takes longer and consumes more energy to overwrite the existing data, which could result in performance degradation and high energy consumption.

2.2. The Impact of Write Latency and Energy

Caches are used to mitigate the performance gap between main memory and the processor. When cache blocks are evicted from the LLC, the dirty blocks will be written back to the main memory. Write requests may generate interference with read requests and therefore affect the performance [Wang et al. 2012]. In addition, the latency and energy overhead associated with the write operation may have significant impacts on the overall system performance/power.

Figure 2 shows average performance and dynamic energy impacts of write requests on various systems for memory-intensive SPEC CPU 2006 benchmarks. Perfect write assumes write requests do not generate any interference with read requests and consume zero dynamic energy, which is the optimal case. We assume that the read and write memory requests for DRAM-based main memory have similar access latency

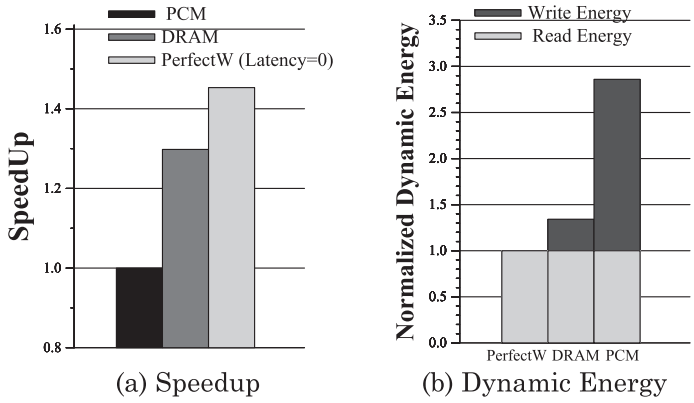

Fig. 2. The performance and dynamic energy impact of write on various systems.

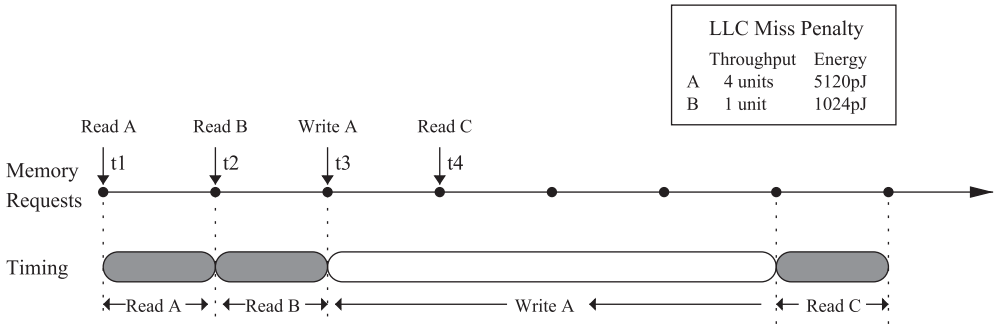

Fig. 3. LLC miss penalty on throughput and energy for dirty cache block and clean cache block.

and dynamic power consumption. For PCM-based main memory, the write latency and energy consumption are assumed to be 10× of that for the read requests. The scheduling policy we used for evaluation is *read prioritizes write* [Wang et al. 2012]. From Figure 2(a), we can see that without write-induced interference, the system performance improves 29.4% in the DRAM-based system and 45.3% in the PCM-based main memory. Figure 2(b) shows the write energy dominates the PCM energy consumption, and it consumes 65% of total dynamic energy consumption, although write requests only account for 25.5% of all the memory accesses. Obviously, both the system performance and the energy efficiency could benefit from reducing the number of writeback requests.

2.3. Motivation

Dirty and clean cache blocks in the LLC have different properties. When dirty cache blocks are evicted from the LLC, they will be written into main memory, incurring performance and energy overhead, while clean cache blocks will not affect the system when they are evicted.

Figure 3 shows an example demonstrating the disparity in LLC miss penalties for dirty data and clean data on PCM throughput and energy. Assuming a request "read A" missed in the LLC and is sent to PCM for service, servicing request "read A" takes one time unit and A is brought into the LLC. Then a request "read B" missed in the LLC and is serviced by PCM for one time unit. In the LLC, "block A" is accessed by a write hit and the dirty bit is set. After "dirty block A" is evicted from the LLC, it will be

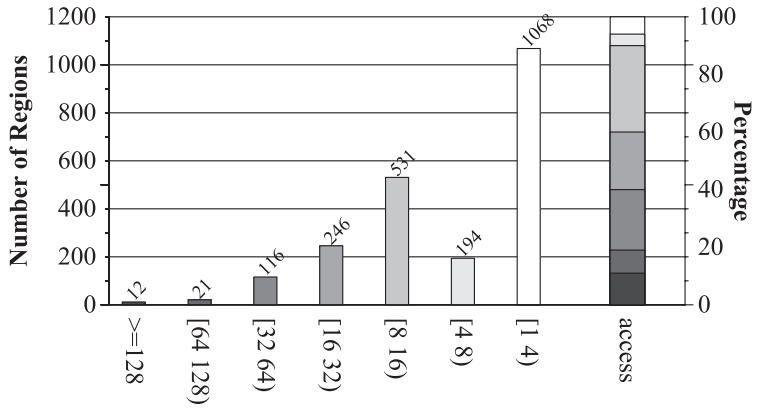

Fig. 4. Region-based memory write access pattern in PCM for *483.xalancbmk* for 500 million instructions. One region contains 16 contiguous blocks. X-axis shows the number of region access times ([*M N*) means the region is accessed by *X* times and $M <= X < N$). Very few regions are accessed frequently (e.g., only 12 regions are accessed more than 128 times).

written back to PCM. Assuming servicing write request "write A" takes four time units. At time $t4$, a request "read C" is sent to PCM that targets the same device with "request A." Then C has to wait until the completion of servicing "A." In this case, C is delayed by servicing request "write A" for three units. Therefore, the LLC performance miss penalty of "clean data B" takes one time unit while the LLC performance miss penalty of "dirty data A" takes four units: one unit for reading "A" and three units for delaying "C." Assuming the PCM read/write energy is 2/8 pJ/bit, the energy miss penalty for "A" and "B" is (64bytes × 8bits × 2pJ/bit) + (64bytes × 8bits × 8pJ/bit) = 5120pJ and 64bytes × 8bits × 2pJ/bit = 1024pJ, respectively. Therefore, the miss penalty for dirty data is more significant than the clean data.

Based on this observation, we propose to adapt the cache management technique to reduce the writeback requests. Since the performance and energy cost are more significant for the dirty cache blocks, the system could benefit by keeping frequent writeback cache blocks in the LLC. However, blindly allocating large cache capacity to frequent writeback data can evict the more critical cache blocks that will be re-referenced soon. This will result in performance degradation. Consequently, there are two questions that need to be answered: *(1) Are the frequent writeback blocks predictable? (2) What is the optimal cache capacity that should be allocated to frequent writeback data?* We performed experiments and have the following two observations:

Observation 1: The writeback accesses have spatial and temporal locality. A small percentage of regions account for a large percentage of writeback accesses. Within a heavily accessed region, the writeback accesses are clustered.

Figure 4 shows access patterns for writeback requests to PCM for the benchmark *xalancbmk* for 500 million instructions. We evaluate the access pattern at the region level. One region includes 1/4 size of the memory page, which has 16 contiguous blocks. The x-axis shows the number of region access times; for example, [32 64) means the region is accessed no less than 32 times and less than 64 times. The y-axis gives the number of regions that correspond to the access times on the x-axis. For instance, the first bar shows there are 12 regions that have been accessed more than 128 times. The last bar shows the percentage of the number of accesses for each type of region that account for all the writeback accesses. We can see the writeback

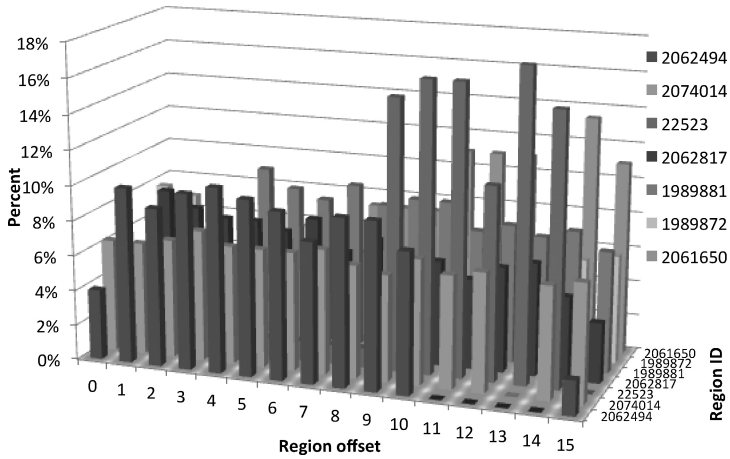

Fig. 5. 3D view for write access pattern in PCM within seven hot regions for *483.xalancbmk*. The x-axis shows the 16 cache blocks within a region. The z-axis shows seven regions that the number of writeback accesses larger than 64.

accesses have temporal locality. Less than 18% percent of regions account for 60% writeback accesses. Figure 5 shows the 3*D* graph for writeback access pattern within the frequent writeback regions. The x-axis shows the 16 cache blocks within a region. The z-axis shows seven regions that the number of writeback accesses larger than 64. The y-axis gives the percentage of total write accesses for each block within the region. We can see the writeback accesses for blocks are clustered within the region.

Based on this observation, we propose a two-stage predictor for frequent writeback cache blocks, at both coarse granularity and fine granularity: The *region granularity* prediction predicts the hot region by capturing the spatial locality and temporal locality. The *cache line granularity* prediction identifies the frequent writeback blocks within the hot region.

Observation 2: The segment size of the frequent writeback list for the cache set significantly affects the performance and energy consumption for workloads.

The last-level cache set is partitioned into *frequent writeback list* and *nonfrequent writeback list*. The *frequent writeback list* consists of frequent writeback cache blocks, while the *nonfrequent writeback list* consists of the remaining cache blocks in the set. Figure 6 shows the performance and energy impact for various sizes of frequent writeback lists for benchmark *perlbench*. For a 16-way LLC, the best segment size for *perlbench* is 11, which generates the best performance and lowest energy cost. We can see that the segment size of the frequent writeback list does significantly affect the performance and energy consumption.

Based on this observation, we propose to segment the cache set into frequent writeback list and nonfrequent writeback list. A segment predictor [Khan et al. 2012] is used to dynamically learn an optimal size of each list in the set according to the miss penalty for dirty and clean cache blocks.

3. WADE: WRITEBACK-AWARE DYNAMIC CACHE MANAGEMENT

The WADE technique improves system efficiency by reducing frequent writes to main memory. Figure 7 shows the structure of WADE. It uses a Frequent Write Predictor (FWP) to predict LLC blocks that are written back to main memory with high frequency

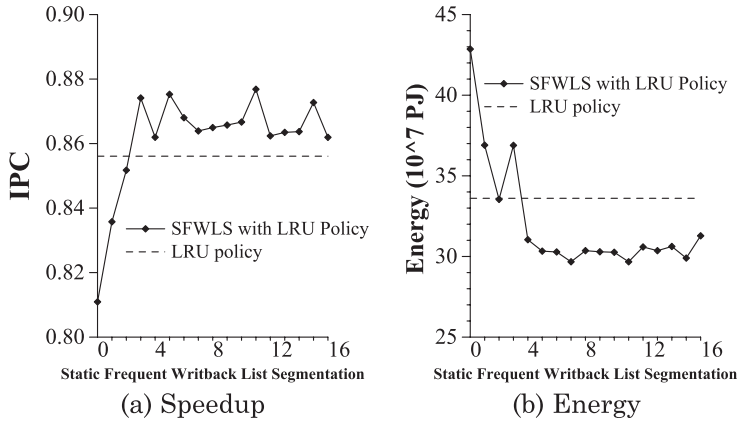

Fig. 6. The impact on performance and energy for various sizes of writeback list for *400.perlbench*. For a 16-way LLC, the optimal segmentation size for frequent writeback list is 11.

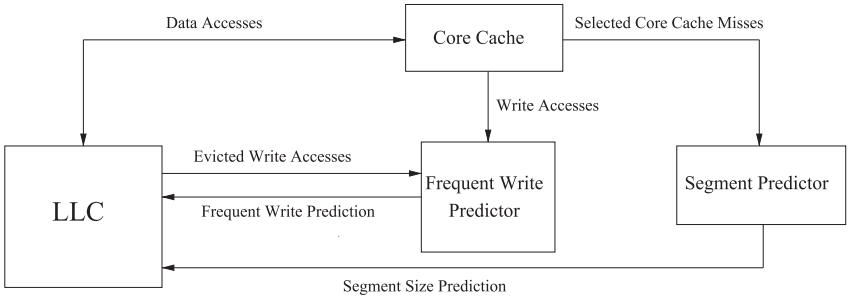

Fig. 7. System structure.

within a certain access interval. The insight of the technique is that frequent writeback data is also highly reused dirty data in the LLC. If frequent writeback data can be stored in the LLC, it can reduce write-induced interference as well as energy consumption of PCM. However, blindly replacing LLC blocks with frequent writeback data can evict more critical cache blocks that have a larger miss penalty, such as clean cache blocks accessed more frequently than the predicted frequent writeback cache blocks. This can lead to performance degradation. In the WADE technique, the LLC set is partitioned into frequent writeback list and nonfrequent writeback list. A segment predictor [Khan et al. 2012] is used to intelligently learn the best partition size of each list.

3.1. Frequent Write Prediction

A frequent write predictor is proposed to keep track of the frequent writeback data and predict the frequent writeback block in the LLC. Figure 8 shows the structure of the FWP, which is located on chip along with the LLC tag arrays. The FWP is organized as a set-associative structure. Every m LLC set maps to n FWP set. Figure 9 shows the address mapping scheme for the FWP. In our experiment, we set $m = 16$, $n = 4$. This address mapping scheme allows the FWP to keep track of the frequent writeback data in region granularity where each region consists of m cache blocks.

Each entry in the FWP set has a partial tag field (PTag), an LRU field, a frequency counter field indicating how often the region data is being written back, and a set flag field in which each flag bit corresponding to each LLC set maps to this FWP set. The set

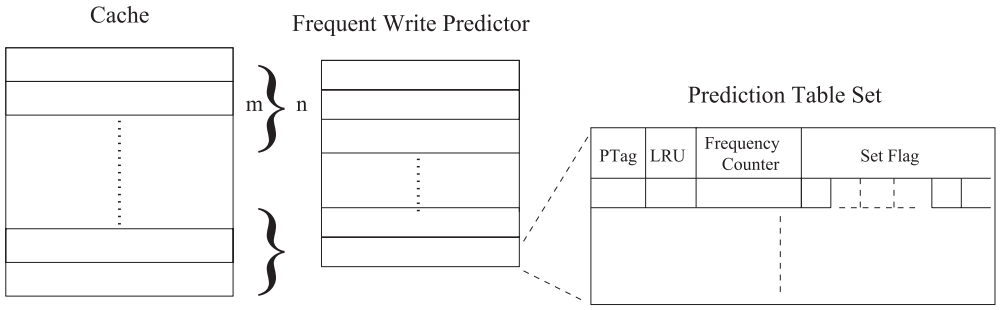

Fig. 8. Illustration of frequent write predictor. FWP is a set-associative structure, and each set has multiple entries with multiple fields.

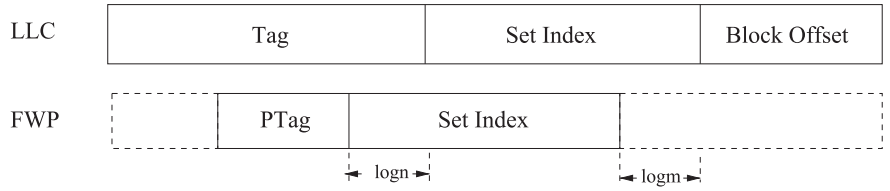

Fig. 9. FWP address mapping scheme. Every m LLC set maps to n FWP.

flag field allows the technique to keep track of frequent writeback data at the cache line granularity. Thus, the FWP table keeps track of the frequent writeback data in both coarse granularity and fine granularity: region granularity and cache line granularity, respectively. Since applications often have spatial and temporal locality, tracking data in coarse granularity (region granularity) can minimize the capacity overhead as well as improve prediction accuracy.

3.1.1. Making a Prediction. For each cache block in the LLC, one Fbit is added, indicating that the block is a frequent writeback block. Once a write request accesses the LLC, it will also access the FWP set for partial tag matching. Since correctness of matches is not necessary in the tag array, only 16 bits of tag are stored in the FWP set entry to conserve area and energy. If it is a partial tag hit and the corresponding set flag bit is set, the Fbit for this cache block is set, indicating that the cache block is a frequent write cache block. Otherwise, the Fbit of the cache block is unset.

3.1.2. Updating Predictor. Once a dirty cache block is evicted from the LLC, the FWP is updated. The evicted dirty cache block accesses the FWP. The LRU recency in the corresponding FWP set is updated for each access. On a partial tag hit, the frequency counter value in the entry is increased by 1. The corresponding set flag bit is set to 1. On a miss, a new entry is allocated in the FWP set. The initial frequency counter value is reset to 0. The corresponding set flag bit is set to 1, while all the other set flag bits in the set flag fields for the newly allocated entry are reset to 0. The replacement candidate is chosen by taking into account both recency and frequency information. The frequency information is used to recognize the frequent writeback region. The recency information can be used to remove the stale data in the FWP table. Assuming the LRU recency value is $R(i)$, where the highest value indicates MRU position and the frequency counter value is $F(i)$, then the replacement victim is chosen as follows:

$$Victim = arg\min_{i}\{F(i) + \gamma R(i)\}. \quad (1)$$

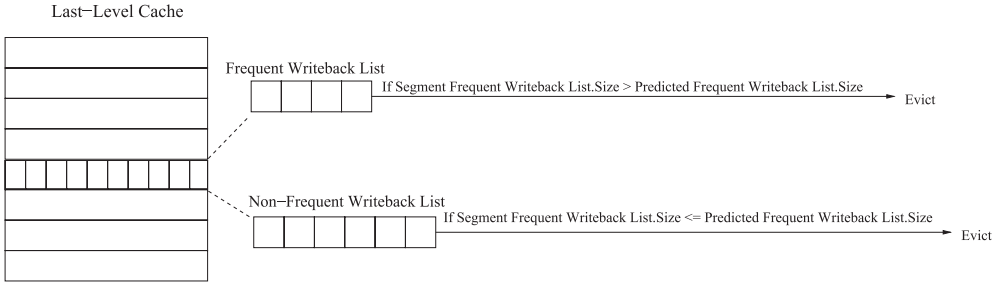

Fig. 10. The logical view of frequent writeback list segmentation mechanism. Each set is partitioned into frequent writeback list and nonfrequent writeback list.

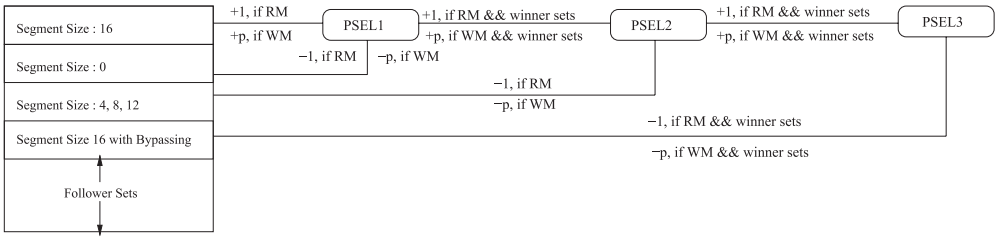

Fig. 11. The mechanism of segment predictor. It consists of six leader sets with segment size 0, 4, 8, 12, 16, and segment size 16 with bypassing.

The parameter γ gives the weight of $R(i)$. It determines the access interval for computing the frequency for writeback data. The larger the value, the smaller the access interval. If the access interval is too small, it could result in a local optimal prediction result instead of a global optimal prediction result. If the access interval is too large, the stale data stored in the FWP prevent the learning process. In our experiment, we found $\gamma = 4$ gives the best performance.

3.2. Frequent Writeback List Cache Segmentation

The LLC set is logically segmented into frequent writeback and nonfrequent writeback lists. The cache blocks with the Fbit set belong to the frequent writeback list; the remaining cache blocks belong to the nonfrequent writeback list. The segment predictor [Khan et al. 2012] is used to predict the optimal segment size of the frequent writeback list for all sets. Figure 10 illustrates the mechanism of the technique. It tries to keep the optimal segment size that minimizes the LLC miss penalties. The technique is decoupled from the LLC replacement policy. Any replacement policy can be applied to each list.

Once a request accesses the LLC, all the ways in the set are searched. On a miss, the size of the frequent writeback list of the set is calculated. If it is larger than the predicted optimal size, the replacement candidate will be chosen from the frequent writeback list. Otherwise, it will be chosen from the nonfrequent writeback list.

3.2.1. Optimal Segment Size Prediction. The segment predictor [Khan et al. 2012] uses set dueling to determine optimal segment size. It estimates the miss penalty for any given segment size by always dedicating a few "leader sets" following that segment size. As shown in Figure 11, we evaluate five segment sizes for a 16-way associative set: 0, 4, 8, 12, 16. The leader sets use decision tree analysis to pairwise set a duel at each level as proposed in Khan et al. [2012]. For instance, segment size 8 duels with segment size 16 in the first level. The policy selection counter 1 (PSEL1) increases on

a miss in leader sets following segment size 8 and decreases on a miss in leader sets following segment size 16. The PSEL1 estimates which segment size is the winner size in the first level. If size 8 is the winner size, the second-level duel will be between segment size 0 and 8. Otherwise, the second-level duel will be between size 12 and 16. The process will continue until the optimal segment size is found. In our experiment, we use an out-of-cache segment predictor, which is a set-associative structure that is added to simulate the sampled leader sets. The LLC sets follow the optimal segment size predicted by the segment predictor.

3.2.2. Bypass Incoming Read Blocks. If a block to be placed in a set will not be reused before it is evicted from the set, it should bypass the cache. Bypassing can improve cache efficiency by allocating the capacity to other reused blocks in the cache. Our segment predictor also considers bypassing the read requests. If the predicted optimal segment size is 16, the leader sets bypassing the read requests duel with the leader sets of segment size 16 without bypassing. The LLC sets will follow the winner policy indicated by PSEL3.

3.2.3. Determining Miss Penalty. The traditional cache replacement policy assumes the absolute number of cache misses is fully correlated with memory-related stall cycles [Qureshi et al. 2006]. It assumes the same miss penalty for dirty and clean cache blocks. In the traditional set dueling technique, for each leader cache set miss, whether the data is dirty or clean, the PSEL is increased/decreased by 1. Our technique is different from previous work in that it is aware of the write inefficiency problem and assigns miss penalty according to the type of cache blocks. If a clean cache block is evicted from the leader set, the PSEL is increased/decreased by 1. If a dirty cache block is evicted from the leader set, the PSEL is increased/decreased by p, defined as follows:

$$p = 1.5 + 0.5 \times l. \tag{2}$$

l is defined as:

$$l = W/R. \tag{3}$$

In the formula, W is the write latency, while R is the read latency. For a certain memory system, l is a constant. Then l is quantized into 2 bits value divided by 8. The larger the value of l, the larger the write latency p. p is measured in steps of 0.5. For each leader set, we add a one-bit even-write flag. If p is not an integer, such as $p = 1.5$, then for every two write misses, the PSEL is increased by three.

4. EVALUATION

In this section, we first outline the experimental methodology used in this study. We then discuss results of our experiments.

4.1. Methodology

We use the MARSSx86 [Patel et al. 2011], a cycle-accurate simulator for the x86-64 architecture. We modify the DRAMSim2 [Rosenfeld et al. 2011] simulator to simulate PCM memory and incorporate it into MARSSx86. The system configuration is shown in Table I. We use the SPEC CPU 2006 [Henning 2006] benchmarks for the evaluation. Each benchmark is run with the first *ref* input provided by the *runspec* command.

Single-Thread Workloads. We use 15 memory-intensive benchmarks for this study. A 2MB LLC is simulated for the single-thread workloads. For each workload, we made a checkpoint by running the benchmark to a typical phase identified by SimPoint

Table I. System Configuration
Memory timing and energies are adapted from Lee et al. [2009]

Execution core	4.8GHZ, 1-core/4-core CMP, out-of-order 256 entry reorder buffer, 4 width issue/decode 15 stages, 256 physical registers
Caches	L1 I/D-cache: 64KB, 2-way, private 64 bytes block, 2-cycle, LRU, L2 Cache: 2MB/1-core, 8MB/4-core 16-way, shared, 64 bytes block, 14-cycle
PCM	1 channel/1-core, 2 channels/4-core CMP 8 banks per channel, 8K bytes row buffer 32-entry write buffer per channel read prioritize write scheduling policy
PCM Timing	row hit (clean miss, dirty miss) = 200 (450, 5000) cycles
PCM Energy	array read (write) = 2.47 (16.82) pJ/bit row buffer read (write) = 0.93 (1.02) pJ/bit

Table II. Workloads

Name	Benchmarks
Mix 1	milc gcc xalancbmk tonto
Mix 2	GemsFDTD namd bzip2 gamess
Mix 3	gamess soplex libquantum perlbench
Mix 4	zeusmp lbm xalancbmk calculix
Mix 5	gamess milc namd soplex
Mix 6	astar lbm gobmk calculix
Mix 7	soplex calculix tonto lbm
Mix 8	lbm mcf cactusADM GemsFDTD
Mix 9	mcf soplex zeusmp bwaves
Mix 10	lbm milc astar libquantum
Mix 11	xalancbmk lbm perlbench tonto

[Sherwood et al. 2002]. Then we run the experiment; starting from the checkpoint, the infrastructure simulates 200 million instructions from the checkpoint.

Multicore Workloads. Table II shows 11 mixes of SPEC CPU 2006 benchmarks chosen four at a time with a variety of memory behaviors. We use these mixes for quad-core simulations. Each benchmark runs simultaneously with the others. For each mix, we made a checkpoint by running one of the memory-intensive benchmarks to a typical phase. Then we run the experiment for 1 billion instructions total for all four cores starting from the checkpoint. We simulate an 8MB shared LLC for the multicore workloads.

4.2. Evaluation Results for Single-Core Workloads

4.2.1. Performance Evaluation. We evaluate three cache replacement polices: the LRU, WADE with LRU, and Memory Level Parallelism (MLP) aware cache replacement techniques [Qureshi et al. 2006]. The MLP technique takes into account the memory-level parallelism-dependent cost differential between different misses. The replacement decision is made by considering the MLP-based cost for each cache miss as well as the recency information. The baseline technique is the LRU replacement policy. Our technique segments the cache set into two lists. Within the list, any replacement policies could be applied. So it is decoupled with LLC replacement policies. We use the

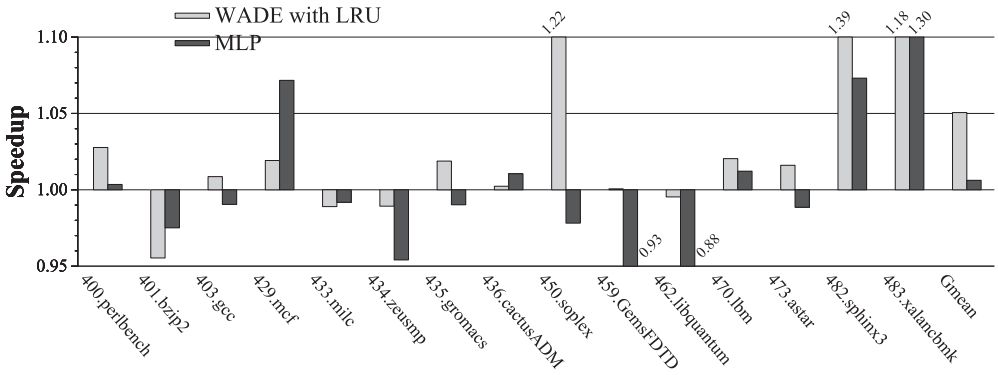

Fig. 12. The comparison of IPC for single-core applications (normalized to LRU).

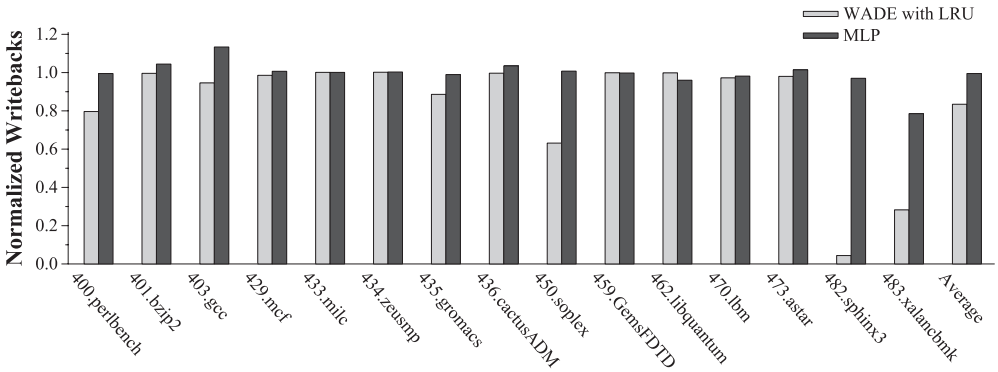

Fig. 13. The number of writeback requests to PCM for single-core applications (normalized to LRU).

LRU replacement policy with our techniques for simplicity. Figure 12 shows the performance evaluation results for single-core applications. MLP provides a speedup on some benchmarks and a slowdown on others, resulting in a geometric mean speedup of approximately 0.6%. The long write latency in the PCM system makes it hard to learn the memory-level parallelism cost; therefore, the MLP replacement policy does not perform well in the context of the PCM system. The WADE technique delivers a geometric mean speedup of 5.1%. The technique significantly improves system performance for benchmarks *450.soplex*, *482.sphinx3,* and *483.xalancbmk* by 22%, 39%, and 18%, respectively. Because of this, the writeback requests for these three benchmarks are highly reused. For benchmarks that do not benefit from our techniques, there are two categories: first, they do not have significantly highly reused access requests, such as for streaming benchmarks *libquantum* and *milc*, so the writeback requests are not rewritten frequently. Second, the frequent writeback requests are hard to predict mainly because they do not have good spatial and temporal locality, such as *436. cactusADM*.

4.2.2. Reduced Write Request Evaluation. The WADE technique takes into account the disparity in miss penalty of clean data and dirty data. It keeps an optimal size of the frequent writeback list in the LLC. Therefore, it can reduce the writeback requests to the PCM. Figure 13 shows the writeback requests normalized to the LRU policy. The MLP technique only reduces 0.05% writeback requests compared with the LRU policy. The WADE technique reduces 16.5% writeback requests on average. This large

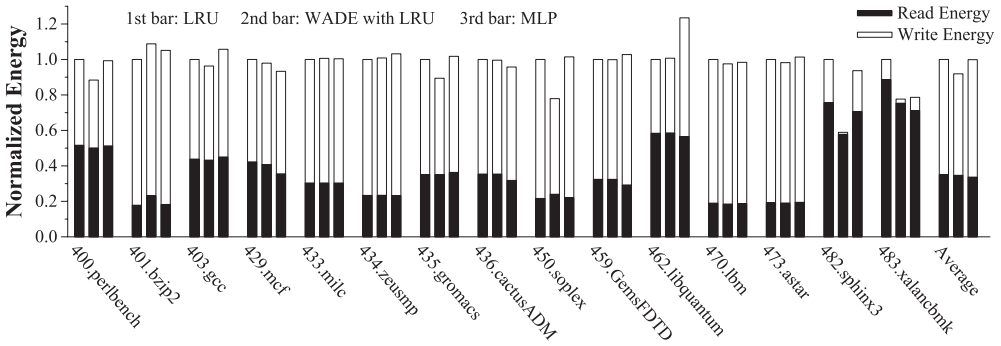

Fig. 14. The comparison of energy consumption in PCM for single-core applications (normalized to LRU).

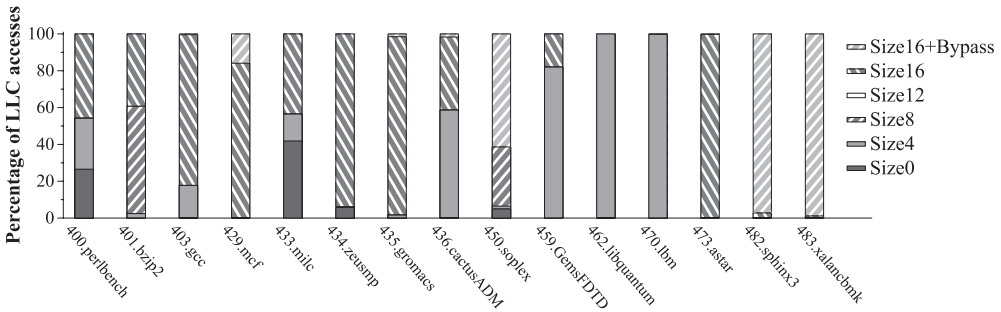

Fig. 15. Runtime-predicted best frequent writeback list size.

percentage of writeback request reduction leads to performance improvement and energy reduction. It can also improve the endurance of the PCM-based main memory. Compared with Figure 12, we can see that the benchmarks that have a large percentage of reduced writeback requests also have significant performance improvements.

4.2.3. Energy Evaluation. The obvious reduction in the writeback requests can lead to reduced energy consumption in PCM-based main memory. Figure 14 shows the energy evaluation results for various techniques. The figure shows the energy consumption normalized to the LRU policy. It also gives the percentage of read energy and write energy consumption for each workload. In PCM-based main memory, the write energy consumption dominates the main memory energy consumption. It accounts for about 65% of all main memory consumption in the LRU policy. The WADE technique achieves an energy reduction of 8.1%, on average. The MLP technique only reduces the energy by 0.01%. We can see that most of the energy reduction of our techniques comes from the write energy reduction. The average read energy consumption for the WADE technique is similar to LRU.

4.2.4. Dynamic Segment Size. Figure 15 shows the runtime-predicted best frequent writeback list size for each of the benchmarks. Benchmarks *483.sphinx3* and *483.xalancbmk* are thrashing workloads that benefit from bypassing incoming read blocks. Segment size 16 dominates the running phase of benchmarks *403.gcc*, *429.mcf*, *434.zeusmp*, *435.gromacs*, and 473.astar. The runtime-predicted best segment size of benchmarks *462.libquantum* and *470.lbm* is 4. The running phases of other benchmarks go through various segment sizes.

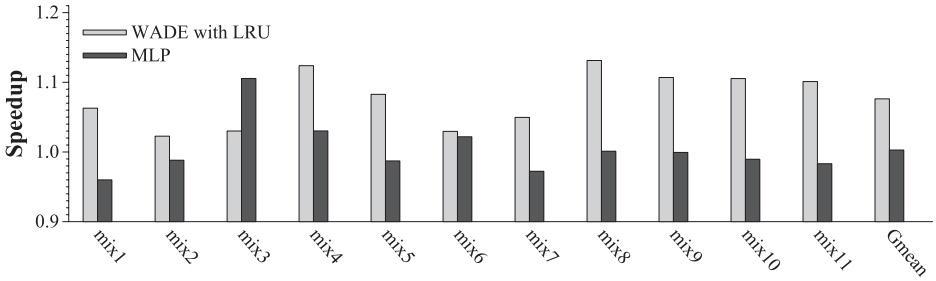

Fig. 16. The comparison of IPC for multicore applications (normalized to LRU).

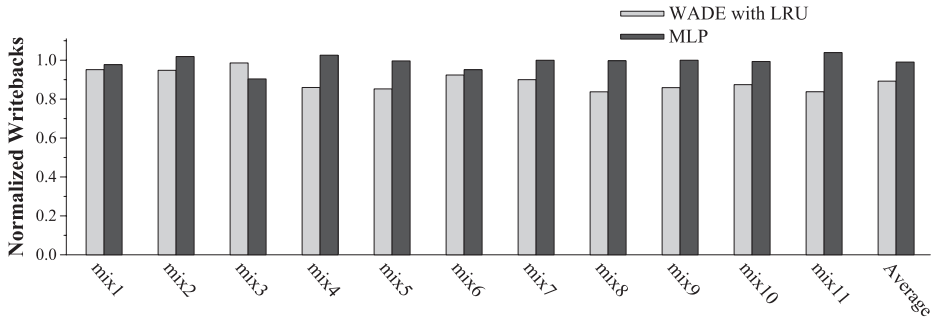

Fig. 17. The number of writeback requests to PCM for multicore applications (normalized to LRU).

4.3. Evaluation Results for Multicore Workloads

The write problem is worse in multicore systems since the performance of an application is affected not only by its own write requests but also by write requests from other applications.

Figure 16 shows the speedup achieved by various techniques on the multicore workloads with an 8MB last-level cache. The speedups are still normalized to a default LRU cache. The normalized speedup for the WADE technique over all 11 workloads ranges from 2.2% to 13.1%, with a geometric mean speedup of 7.6%. The technique significantly improves the system performance for five workloads by more than 10%. The MLP technique only yields a geometric mean speedup of 0.3%.

Figure 17 shows the normalized writeback request evaluation results for multicore applications. The WADE technique achieves a writeback request reduction of 10.9% on average. Figure 18 shows the energy evaluation results normalized to the LRU policy. The WADE technique reduces energy by 7.6%, on average.

We also evaluate the Misses Per 1,000 Instructions (MPKI) for multicore workloads. Figure 19 shows the MPKI for various techniques normalized to the LRU policy. The average normalized MPKIs are 1.00 for WADE and 0.99 for MLP. We can see that the WADE technique does not reduce the miss rate. In the WADE technique, the performance benefits actually come from the reduced write requests, which generate a large write-induced interference.

4.4. Sensitivity Study

4.4.1. Miss Penalty Sensitivity Study. An LLC miss for a dirty cache block is more harmful than for a clean cache block. Our technique assigns different miss penalties according to the type of data. The miss penalty for clean data is set to 1, while the miss penalty for dirty data is p. In our experiment setting, we get $p = 2$ calculated by Equation (2). We

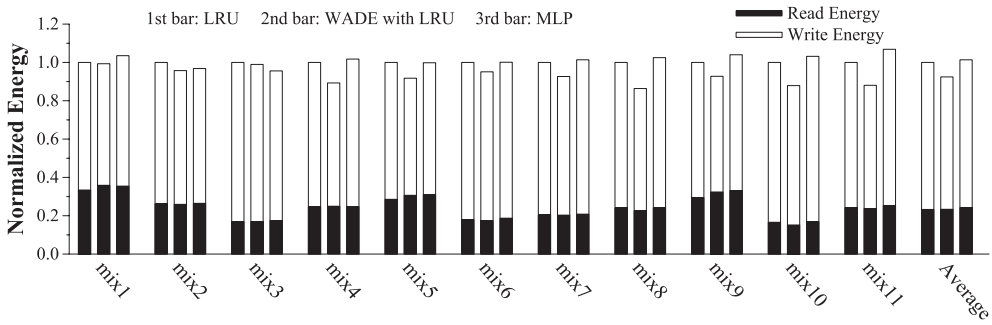

Fig. 18. The comparison of energy consumption in PCM for multicore applications (normalized to LRU).

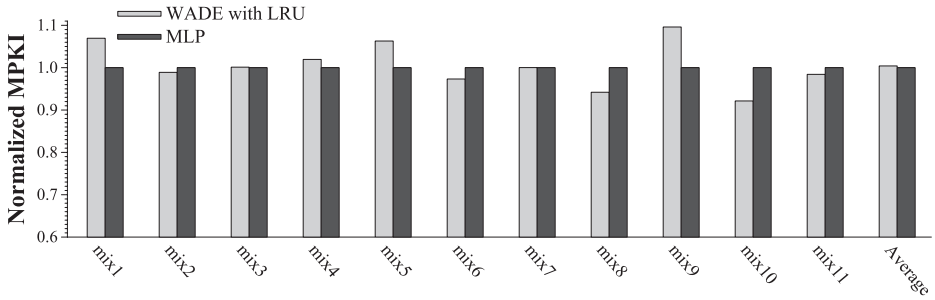

Fig. 19. LLC misses per kilo-instruction (MPKI) for multicore applications (normalized to LRU).

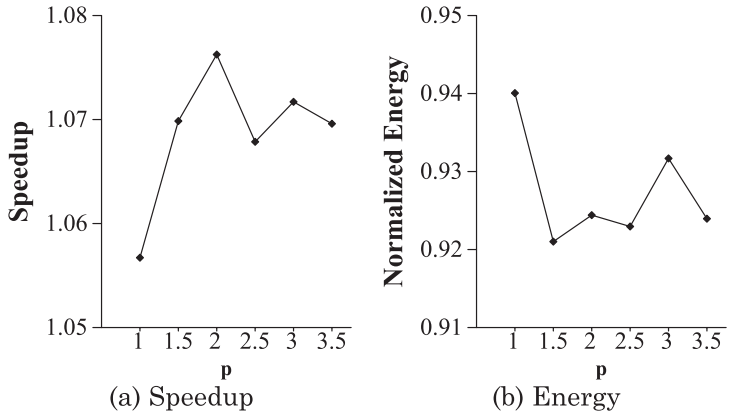

Fig. 20. The impact on performance and energy for parameter p.

also did an experiment to test the change in performance and energy when p ranges from 1 to 6. Figure 20 shows the performance speedup and energy consumption for various values of p in multicore workloads using the WADE technique. We can see the performance and energy consumption varies significantly with different values of p. The best performance is achieved when $p = 2$, and the lowest energy consumption when $p = 1.5$. Generally, the p value that gives better performance is also the value that yields lower energy, because the reduced write requests could lead to both performance improvement and energy reduction. In our experiment, we choose $p = 2$.

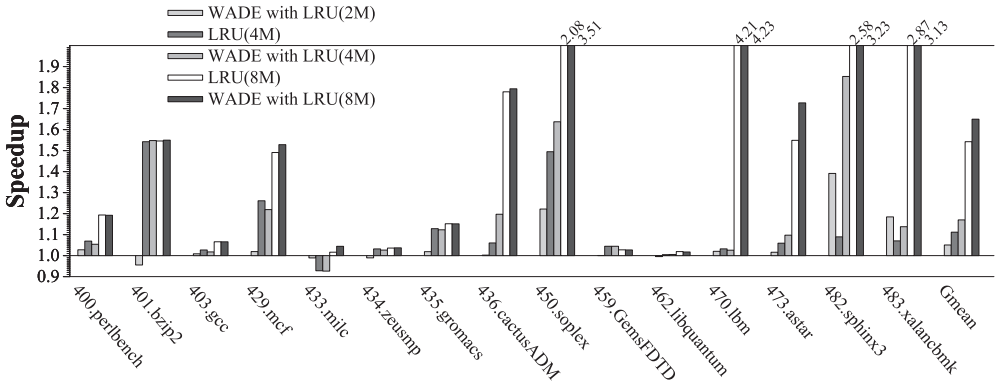

Fig. 21. Performance evaluation with various cache sizes (normalized to LRU with 2M LLC size).

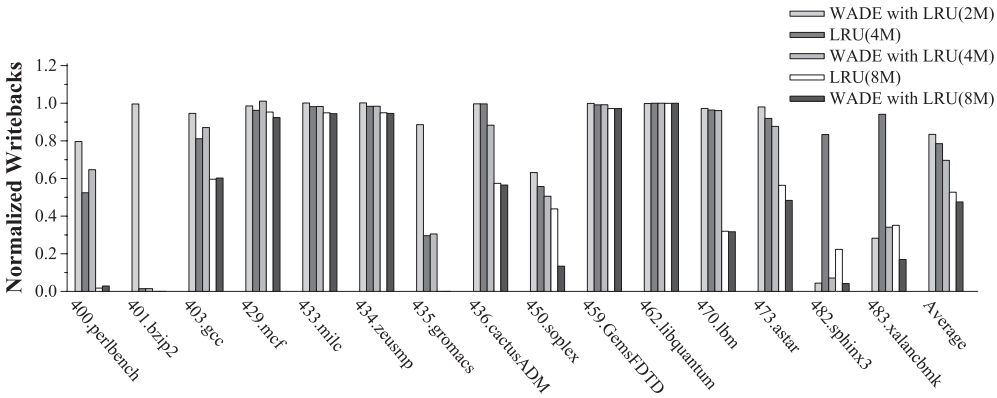

Fig. 22. The number of writeback requests to PCM with various cache sizes (normalized to LRU with 2M LLC size).

4.4.2. Cache Size Sensitivity Study. Figures 21 and 22 show the performance and writeback reduction evaluation results with various cache sizes. We evaluate LRU and WADE LLC replacement policies with cache sizes 2M, 4M, and 8M. Compared with the LRU replacement policy with the same capacity 2M, 4M, and 8M cache sizes, the WADE technique improves the system performance by 5.1%, 5.4%, and 6.9% and reduces the writeback requests to PCM by 16.5%, 11.4%, and 9.9%, respectively.

4.5. Storage and Power Overhead

4.5.1. Storage Overhead. The technique uses an FWP and an optimal segment predictor. For the FWP, every 16 LLC sets map to four FWP sets. Each FWP set has six entries. Each entry in the set has a 16-bit partial tag field, a 3-bit LRU field, a 6-bit frequent counter field, and a 16-bit set flag field. For each cache block in the LLC, we add 1 bit to represent whether it is a frequent writeback block. The FWP consumes an extra state equivalent to about 0.95% of LLC capacity. We use an out-of-cache segment predictor. This set-associative structure is added to simulate sampled leader sets. It uses four types of leader sets, as shown in Figure 11. For each type of leader set, one set is sampled for every 128 LLC sets. Each leader set has a one-bit even-write counter. Each entry in the leader set has a 16-bit partial tag field, a 1 Fbit field, and 3-bit LRU fields. The segment predictor uses three 12-bit PSEL counters. Thus, it consumes less

than 0.13% of LLC capacity. Altogether, the WADE technique takes about 1% of LLC capacity.

4.5.2. Power Overhead. We use CACTI [HP-Laboratories 2008] to measure the potential impact of the segment predictor and frequent write predictor on power. The segment predictor is modeled as a tag array of extra LLC sets. We model the LLC both with and without the extra cache sets and report the difference of the tag power between the two. We model the frequent write predictor as a tag array of a cache, with only the tag power being reported. A 2MB LLC in a single-core configuration consumes $1.99W$ power. The segment predictor consumes only $0.0025W$ dynamic power, which is only 0.13% of LLC power consumption. The power for the frequent write predictor is $0.024W$. The total power for structures required by the WADE technique is about 1.3% of LLC power. An 8M LLC in a multicore configuration consumes $3.73W$. The structures needed by the WADE technique take $0.035W$, which is 0.93% of LLC power. Although the segment predictor and frequent write predictor consume extra power, the WADE technique reduces the execution cycles of applications, thus reducing the leakage energy of the LLC.

5. RELATED WORK

In this section, we summarize related prior works, which can be classified into three categories.

5.1. Related Work on Mitigating PCM Write Overhead

Many researchers propose techniques to mitigate PCM write latency and energy overhead. For example, Lee et al. [2009] propose to use narrow PCM buffers to mitigate high-energy PCM writes. Write cancellation and write pausing [Qureshi et al. 2010] have been proposed to prioritize read requests over write requests by adaptively canceling or pausing the service of write requests when read requests are waiting for service. Qureshi et al. [2012] exploit asymmetry in write times for SET and PRESET operations of PCM devices and propose to initiate a PreSET request for a memory line as soon as data is written into the LLC, thereby incurring low write-induced interference.

Hybrid main memory architecture has been proposed to leverage the benefits of both DRAM and PCM technologies. Qureshi et al. [2009] propose a main memory system consisting of PCM storage coupled with a DRAM write buffer so that it has the latency benefits of DRAM and the capacity benefits of PCM. Yoon et al. [2012] propose to improve the hybrid performance by caching the frequent row buffer miss requests in DRAM. Ramos et al. [2011] propose a page-ranking and migration policy for the hybrid PCM- and DRAM-based main memory.

Write endurance poses another severe challenge in PCM memory design. The cells suffering from more frequent write operations will fail far sooner than the rest. A read-before-write operation [Joo et al. 2010] can help identify such redundant bits and cancel those redundant write operations to save energy and reduce impact on performance. A range of *wear-leveling techniques* [Qureshi et al. 2009; Zhou et al. 2009; Lee et al. 2009] for PCM have been examined to increase the lifetime of PCM-based main memory architectures.

Most of these proposed techniques mitigate the write overhead of PCM by doing optimizations at the main memory level. They either use new memory architectures or add a new operation to PCM. However, write requests sent from the LLC remain unchanged. Our techniques can be combined with these techniques to achieve further improvement.

5.2. Related Work in LLC Writeback Techniques

LLC writeback techniques [Lee et al. 2000; Stuecheli et al. 2010; Wang et al. 2012] have been proposed in the literature. These techniques coordinate the LLC with main memory. They aim to increase the visibility of the memory controller by the LLC, thereby making an optimal writeback scheduling decision. The virtual write queue (VWQ) [Stuecheli et al. 2010] technique takes a fraction of the LRU positions in the LLC as the virtual write queue. Dirty cache blocks in the VWQ that target the same row buffer when mapping to the memory resource will be written back in a batch, therefore reducing write-induced interference. Wang et al. [2012] propose a last-write prediction-driven LLC writeback technique. The technique predicts the last write cache blocks in the LLC. The row buffer hitting the last-write cache blocks are sent to memory for service to improve the memory efficiency. This group of work allows memory to service write requests efficiently; it also does not reduce the write requests serviced by main memory.

5.3. Related Work in Cache Management Policy

Most of the previously proposed cache management policies improve cache efficiency by learning the access pattern, selecting the management policy, or partitioning the cache capacity.

LRU replacement predicts that a block will be referenced in the near future. Re-Reference Interval Prediction (RRIP) [Jaleel et al. 2010] predicts the re-reference interval of the cache block. It prevents blocks with distant re-reference intervals from evicting blocks that have a near re-reference interval. Dynamic Insertion Policy (DIP) [Qureshi et al. 2007] uses set dueling to adaptively select the replacement candidate in either the MRU or LRU position depending on which policy gives better performance.

The MLP [Qureshi et al. 2006] aware cache replacement technique is aware of the MLP-dependent cost differential between different misses. The replacement decision is made by taking into account the MLP-based cost for each cache miss. The Dynamic Cache Segmentation (DCS) [Khan et al. 2012] work segments the cache set into a reused list and a nonreused list and dynamically learns the best size of the segmentation size. Zhou et al. [2012] propose to partition the LLC among multiple applications by considering both LLC misses and writebacks. However, their technique only can be applied to the LRU replacement policy.

Many previous works [Jiang et al. 2011; Cantin et al. 2006; Wu et al. 2011; Cantin et al. 2006; Sudan et al. 2010] propose a cache management policy by using coarse granularity. Cantin et al. [2006] use regions to avoid broadcasts for nonshared regions to reduce bandwidth for a snoop-based cache coherence policy. Wu et al. [2011] propose to use memory region information to learn the cache access behavior in the cache replacement policy.

Our technique is orthogonal to most previous works that either mitigate the write overhead by exploring the PCM property in the main memory level [Qureshi et al. 2009; Zhou et al. 2009; Lee et al. 2009] or optimize the write scheduling policy, such as virtual write queue and last-write prediction. It can be combined with others to further achieve better performance and energy efficiency.

6. CONCLUSION

In this article, we propose a dynamic cache management policy in the context of NVM-based main memory. The technique improves system performance and energy efficiency by reducing the writeback requests to PCM. It keeps highly reused dirty cache blocks in the LLC. A frequent write predictor is proposed to predict the frequent writeback cache blocks. The cache set is partitioned into frequent writeback and nonfrequent

writeback lists. It dynamically determines the optimal size of each list according to the miss penalty. Our evaluation shows that the proposed techniques reduce the writeback requests, which could result in improved performance as well as reduced energy consumption. Most prior work focuses on optimization techniques for NVM-based main memory itself. Our technique is orthogonal to those previous techniques. It can be combined with others to further achieve better results.

REFERENCES

Cantin, J. F., Lipasti, M. H., and Smith, J. E. 2006. Stealth prefetching. *SIGOPS Oper. Syst. Rev. 40,* 5, 274–282.

Cantin, J. F., Smith, J. E., Lipasti, M. H., Moshovos, A., and Falsafi, B. 2006. Coarse-grain coherence tracking: Regionscout and region coherence arrays. *IEEE Micro 26,* 1, 70–79.

Chang, J. and Sohi, G. S. 2007. Cooperative cache partitioning for chip multiprocessors. In *Proceedings of the 21st Annual International Conference on Supercomputing (ICS'07).* ACM, New York, NY, 242–252.

Choi, Y., et al. 2012. A 20nm 1.8v 8Gb PRAM with 40MB/s program bandwidth. In *Proceedings of the IEEE International Solid-State Circuits Conference*.

Hanzawa, S., Kitai, N., Osada, K., Kotabe, A., Matsui, Y., et al. 2007. A 512kB embedded phase change memory with 416kB/s write throughput at 100μA cell write current. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'07).* 474–616.

Henning, J. L. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News 34*, 1–17.

HP-Laboratories. 2008. Cacti 5.3. Retrieved from http://quid.hpl.hp.com:9081/cacti.

Jaleel, A., Hasenplaugh, W., Qureshi, M. K., Sebot, J., Jr., S. S., and Emer, J. 2008. Adaptive insertion policies for managing shared caches. In *Proceedings of the 2008 International Conference on Parallel Architectures and Compiler Techniques (PACT'08)*.

Jaleel, A., Theobald, K., Jr., S. S., and Emer, J. 2010. High performance cache replacement using re-reference interval prediction. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*.

Jiang, X., Madan, N., Zhao, L., Upton, M., Iyer, R., Makineni, S., Newell, D., Solihin, Y., and Balasubramonian, R. 2011. Chop: Integrating dram caches for CMP server platforms. *IEEE Micro 31,* 1, 99–108.

Joo, Y., Niu, D., Dong, X., Sun, G., Chang, N., and Xie, Y. 2010. Energy- and endurance-aware design of phase change memory caches. In *Proceedings of Design, Automation and Test in Europe (DATE'10)*. 136–141.

Khan, S. M., Wang, Z., and Jimenez, D. A. 2012. Decoupled dynamic cache segmentation. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA'12).* IEEE Computer Society, Washington, DC, 1–12.

Lee, B. C., Ipek, E., Mutlu, O., and Burger, D. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09).* ACM, New York, NY, 2–13.

Lee, C. J., Narasiman, V., Ebrahimi, E., Mutlu, O., and Patt, Y. N. 2010. DRAM-aware last level cache writeback: Reducing write-caused interference in memory system. HPS Technical Report.

Lee, H.-H. S., Tyson, G. S., and Farrens, M. K. 2000. Eager writeback—a technique for improving bandwidth utilization. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'00).* ACM, New York, NY, 11–21.

Patel, A., Afram, F., Chen, S., and Ghose, K. 2011. MARSSx86: A full system simulator for x86 CPUs. In *Proceedings of the 2011 Design Automation Conference*.

Pellizzer, F., Pirovano, A., Ottogalli, F., Magistretti, M., Scaravaggi, M., et al. 2004. Novel μTrench phase-change memory cell for embedded and stand-alone non-volatile memory applications. In *Proceedings of the 2004 Symposium on VLSI Technology.* 18–19.

Qureshi, M. K., Franceschini, M. M., Jagmohan, A., and Lastras, L. A. 2012. Preset: Improving performance of phase change memories by exploiting asymmetry in write times. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA'12).* IEEE Press, Piscataway, NJ, 380–391.

Qureshi, M. K., Franceschini, M. M., and Lastras-Montao, L. A. 2010. Improving read performance of phase change memories via write cancellation and write pausing. In *International Symposium on High Performance Computer Architecture* (*HPCA'10).* 1–11.

Qureshi, M. K., Jaleel, A., Patt, Y. N., Steely, S. C., and Emer, J. 2007. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07).* ACM, New York, NY, 381–391.

Qureshi, M. K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., and Abali, B. 2009. 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 (MICRO'09). ACM, New York, NY, 14–23.

Qureshi, M. K., Lynch, D. N., Mutlu, O., and Patt, Y. N. 2006. A case for mlp-aware cache replacement. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA'06)*. IEEE Computer Society, Washington, DC, 167–178.

Qureshi, M. K. and Patt, Y. N. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE Computer Society, Washington, DC, 423–432.

Qureshi, M. K., Srinivasan, V., and Rivers, J. A. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*.

Ramos, L. E., Gorbatov, E., and Bianchini, R. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing (ICS'11)*. ACM, New York, NY, 85–95.

Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y.-C., et al. 2008. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development 52,* 4/5.

Rosenfeld, P., Cooper-Balis, E., and Jacob, B. 2011. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters PP,* 99, 1.

Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*.

Stuecheli, J., Kaseridis, D., Daly, D., Hunter, H. C., and John, L. K. 2010. The virtual write queue: coordinating dram and last-level cache policies. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY, 72–82.

Sudan, K., Chatterjee, N., Nellans, D., Awasthi, M., Balasubramonian, R., and Davis, A. 2010. Micro-pages: Increasing DRAM efficiency with locality-aware data placement. In *Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)*. ACM, New York, NY, 219–230.

Sun, G., Dong, X., Xie, Y., Li, J., and Chen, Y. 2009. A novel architecture of the 3d stacked mRAM L2 cache for CMPs. In *HPCA*. 239–249.

Wang, Z., Khan, S. M., and Jiménez, D. A. 2012. Improving writeback efficiency with decoupled last-write prediction. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA'12)*. IEEE Press, Piscataway, NJ, 309–320.

Wu, C.-J., Jaleel, A., Hasenplaugh, W., Martonosi, M., Steely, Jr., S. C., and Emer, J. 2011. Ship: signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. ACM, New York, NY, 430–441.

Xie, Y. 2011. Modeling, architecture, and applications for emerging memory technologies. *IEEE Computer Design and Test, 28,* 41–51.

Xie, Y. and Loh, G. H. 2009. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 174–183.

Yoon, H., Meza, J., Ausavarungnirun, R., Harding, R., and Mutlu, O. 2012. Row buffer locality aware caching policies for hybrid memories. In *Proceedings of the International Conference on Computer Design (ICCD'12)*.

Zhou, M., Du, Y., Childers, B., Melhem, R., and Mossé, D. 2012. Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems. *ACM Trans. Archit. Code Optim. 8,* 4, 53:1–53:21.

Zhou, P., Zhao, B., Yang, J., and Zhang, Y. 2009. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 14–23.

Received June 2013; revised August 2013; accepted November 2013