Segment and Conflict Aware Page Allocation and Migration in DRAM-PCM Hybrid Main Memory

Hoda Aghaei Khouzani, *Student Member, IEEE*, Fateme S. Hosseini, and Chengmo Yang, *Member, IEEE*

***Abstract*—Phase change memory (PCM), given its nonvolatility, potential high density, and low standby power, is a promising candidate to be used as main memory in next generation computer systems. However, to hide its shortcomings of limited endurance and slow write performance, state-of-the-art solutions tend to construct a dynamic RAM (DRAM)-PCM hybrid memory and place write-intensive pages in DRAM. While existing optimizations to this hybrid architecture focus on tuning DRAM configurations to reduce the number of write operations to PCM, this paper explores the interactions between DRAM and PCM to improve both the performance and the endurance of a DRAM-PCM hybrid main memory. Specifically, it exploits the flexibility of mapping virtual pages to physical pages, and develops a proactive strategy to allocate pages taking both program segments and DRAM conflict misses into consideration, thus distributing those heavily written pages across different DRAM sets. Meanwhile, a lifetime-aware DRAM replacement algorithm and a conflict-aware page remapping strategy are proposed to further reduce DRAM misses and PCM writes. Experiments confirm that the proposed techniques are able to improve average memory hit time and reduce maximum PCM write counts thus enhancing both performance and lifetime of a DRAM-PCM hybrid main memory.**

***Index Terms*—Hybrid main memory architecture, nonvolatile memory technologies, page allocation.**

I. Introduction

As SEMICONDUCTOR technology keeps scaling down, the need for a denser, low-power and more reliable main memory becomes more severe. Existing DRAM based main memories not only consume significant amounts of energy but also suffer from limited scalability below 20 nm [1]. These shortcomings have motivated the search for alternatives to replace DRAM as main memory. Phase change memory (PCM), which has better scalability and lower standby energy than DRAM [2], [3], is a promising candidate. However, the limited write endurance, high write energy, and long write latency of PCM challenge its complete replacement to DRAM as main memory. To hide these shortcomings of PCM, one attractive solution is a *DRAM-PCM hybrid architecture*.

DRAM-PCM hybrid memory architectures offer the merit of combining the advantages of DRAM and PCM and hiding their shortcomings. Specifically, as write operations are critical to the performance, energy, and endurance of PCM [2], [3], it is preferable to place pages with more read operations in PCM while storing write-intensive pages in DRAM [4]. Usually two types of hybrid architectures can be adopted. The first one organizes DRAM and PCM in parallel and places a page exclusively either in PCM or in DRAM [5]–[7]. Since in these architectures the size of DRAM is in the same order as the size of PCM, the scalability limitation and the high standby power of DRAM are still dominant. The alternative is to place DRAM and PCM hierarchically and uses DRAM as a small cache for PCM [8]–[10]. These architectures are more efficient, since a DRAM of 3% total memory size is about to deliver similar performance as a DRAM-only system and absorb most of memory writes [11], thus largely reducing the energy-expensive and lifetime-hurting PCM writes. However, a small DRAM size results in a relatively higher DRAM miss rate, which in turn hurts the performance and endurance of hybrid memory since upon a DRAM miss, PCM is accessed, and pages need to be migrated between the two devices.

While most of existing optimizations on DRAM-PCM hybrid memory focus on improving PCM endurance by reducing the number of PCM writes in PCM [10], the goal of this paper is to improve both the performance and the endurance of the hybrid memory at the same time, through directly reducing the number of DRAM misses and the resultant writebacks to PCM. However, the straightforward solutions such as enlarging the DRAM size or increasing the way-associativity of the DRAM cache are not efficient. The former increases the standby power and the footprint of the main memory, while the latter complicates the process of tag matching and hence increases the latency and energy of each DRAM access.

To reduce DRAM misses without imposing too much overhead or sacrificing memory performance, this paper presents a proactive solution that exploits the flexibility of mapping virtual pages to physical pages. Specifically, the page allocation process is tuned to consider segment and conflict information.

1) The proposed technique leverages program segment information to differentiate write-intensive and read-only pages. By placing read-only pages in PCM and furthermore bypassing DRAM when accessing those pages, *capacity misses* in DRAM can be reduced.
2) As memory accesses of most programs are highly clustered, a high variation in the number of conflict misses

Manuscript received April 8, 2016; revised August 10, 2016; accepted September 21, 2016. Date of publication October 7, 2016; date of current version August 18, 2017. This paper was recommended by Associate Editor S. Pasricha.

The authors are with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716 USA (e-mail: hoda@udel.edu; fateme@udel.edu; chengmo@udel.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2615845

across different DRAM sets is expected. This paper develops a novel hierarchical clock algorithm, capable of allocating pages to less-conflicting DRAM sets, thus effectively reducing the *conflict misses* in DRAM.

The segment and conflict-aware page allocation algorithm was originally proposed in [12]. This paper enhances it in several new directions. First, a detailed analysis of performance, energy, and endurance of the DRAM-PCM hybrid main memory is conducted to illustrate the necessity for reducing DRAM misses. Moreover, two new algorithms are proposed: one remaps heavily-conflicting pages when DRAM misses do not display repetitive behavior, while the other selects eviction targets in DRAM based on PCM write counts. Finally, extensive experimental studies are conducted to evaluate the proposed algorithm and compare it with the state-of-the-art solutions.

The rest of this paper is organized as follows. Section II provides background knowledge of PCM and related works. Section III illustrates the impact of DRAM misses on the performance, energy, and endurance of the hybrid main memory. Section IV introduces the proposed hybrid main memory architecture along with its access policy, and describes the proposed page allocation, remapping, and replacement algorithms. Section V presents results of our experimental studies, while Section VI concludes this paper.

II. Background and Related Work

A. Phase Change Memory

PCM is made of chalcogenide glass, a phase-change material with two states: 1) amorphous and 2) crystalline. The two states have large resistance contrast, which is exploited to represent 1 and 0, or even multiple states per cell. Transitions between these states are driven by heating processes [2]: heating a cell above the crystallization temperature (300 °C) for a certain period of time transforms it into the crystalline state, while heating a cell above the melting temperature (600 °C) and quenching quickly transforms it into the amorphous state.

Table I summarizes the attributes of PCM and DRAM [1]. As a volatile memory, DRAM requires consistent refresh cycles. It also suffers from limited scalability below 20 nm due to the constraints of leakage current and capacitor placement [1]. PCM, on the other hand, does not require any power-hungry refresh cycle since its states are persistent. It is also expected to have better scalability than DRAM. A PCM device prototype with the footprint of 3×20 nm is reported in [2]. Reads in PCM consume small amount of energy, while writes are long and power hungry due to the need of a heating process to change PCM states. More crucially, repetitive heating stress will cause the material to stick at either the crystalline or the amorphous state permanently, thus imposing a constraint of 10^6–10^9 writes that a PCM cell can sustain. Compared to the typical 10^{16} writes that a DRAM cell can sustain, such limited programming cycles largely constrain the lifetime of PCM-based main memory.

B. PCM-Based Main Memory

Researchers have examined the possibility of using PCM exclusively as main memory. These works tackle the limited write endurance of PCM from three perspectives. The first set of works aims at reducing the total number of bit flips applied to PCM cells [3], [11], [13], [14]. Upon each write request, they first compare the value to be written with the old value, and then rewrite only those different bits. The second group of works tries to tackle the imbalanced write distribution caused by the spatial and temporal localities inherent in applications. To prevent the undesired situation of a small set of cells worn out quicker than the others, these wear-leveling techniques constantly remap hot data to cold regions at various granularities [15]–[20]. The last set of techniques does not prevent cells from becoming worn out. Instead, they treat worn-out cells as permanent, stuck-at errors, and apply error detection and/or correction techniques [21]–[23] to tolerate them.

TABLE I
Comparison of PCM and DRAM

Attributes	PCM	DRAM
Scalability	45nm	65nm
Read Latency	∼10ns	∼10ns
Write Latency	∼100ns	∼10ns
Read Energy (pJ/bit)	2.5	4.4
Write Energy (pJ/bit)	14(set)∼20(reset)	5.5
Endurance (Write Cycles)	$10^6 \sim 10^9$	10^{16}

In addition to its highly constrained write endurance, PCM also suffers from high write energy and long write latency. To simultaneously address all these issues, one solution is to not use PCM exclusively, but organize PCM and DRAM in a hybrid memory architecture so as to exploit the advantages of both devices and to overcome their drawbacks.

C. DRAM-PCM Hybrid Main Memory

The designs which combine PCM and DRAM in hybrid main memory architectures can be divided into two main categories, as shown in Fig. 1. The first one organizes DRAM and PCM in *parallel* with the size of DRAM being in the same order as the size of the PCM, while the second one places DRAM and PCM *hierarchically* and uses DRAM as a small cache for PCM. In both scenarios, the underlying principle is to place write-intensive pages in DRAM and write-infrequent pages in PCM. To reduce runtime overhead, OS is not directly involved in the decision of migrating pages between the two devices. Instead, a memory controller (MC) usually manages the traffic between PCM and DRAM.

When DRAM and PCM are organized in parallel, the entire address space is split into two nonoverlapping regions, as shown in Fig. 1(a). A page is exclusively stored in one of the two devices. Research targeting this organization mainly focuses on the identification of hot data in PCM and cold data in DRAM for migration. One of the earliest parallel DRAM-PCM structures is PDRAM [5] which adds a counter to each memory page to record its write frequency, and uses such information to trigger page migrations. However, this method is prone to generate unnecessary writes to PCM. In [6], a memory access monitoring technique that periodically scans two reference bits in the page table to distinguish hot/cold pages is presented. Hot/cold pages are periodically migrated to DRAM/PCM to reduce overall energy consumption. Lee *et al.* [7] concerned about the slow write performance of PCM and propose a page replacement

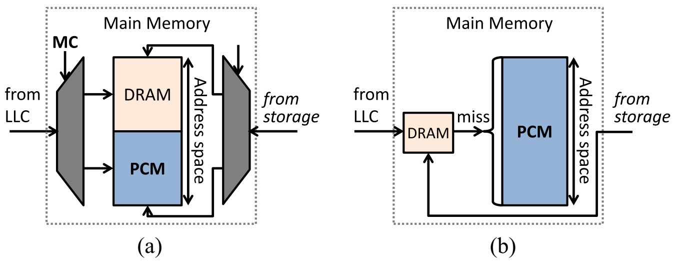

Fig. 1. Two organizations of DRAM-PCM hybrid memory that differ in address space, DRAM size, and handling accesses from LLC. (a) Parallel organization. (b) Hierarchical organization.

algorithm called CLOCK-DWF. If a memory write causes a page fault, CLOCK-DWF places that page in DRAM and searches for a clean page in DRAM to be migrated to PCM. Another work [24] uses *row buffer locality* to guide data placement. Row buffers store the most-recently-accessed rows in both DRAM and PCM. To reduce the average memory access latency, data that frequently miss in the row buffer are placed in DRAM. In [25], segment information is exploited to guide data placement. The code segment is allocated in PCM, the stack and global data segments are placed in DRAM, and the heap is dynamically migrated between the two devices based on the read/write ratio. Recently a software enabled wear-leveling technique is proposed in [26] to extend PCM lifetime. It assigns the variables with the highest number of writes to DRAM, and distributes the other variables to different regions in PCM based on the number of writes to those regions.

In the hierarchical organization wherein DRAM is used as a cache for PCM, all the memory accesses are directed to DRAM while PCM is accessed only upon DRAM misses, as shown in Fig. 1(b). DRAM size is usually small (approximately 3% of PCM size) [11] and may be in the same order as the size of the last level cache (LLC). However accesses missed in the LLC may still hit in DRAM since the LLC contents are not forced to be "inclusive" in DRAM.

Most of existing research targeting the hierarchical organization focuses on reducing the number of PCM writes. Qureshi *et al.* [11] developed one of the earliest works to take advantage of the high density of PCM and the high performance of DRAM, which is used as the baseline by several later optimizations. For instance, Ferreira *et al.* [10] enhanced the baseline hierarchical DRAM-PCM organization by further reducing PCM writes. They proposed an *N-chance* cache replacement policy that gives a higher priority to clean blocks in DRAM for eviction. This technique reduces PCM writes at the cost of higher DRAM miss rates and degraded memory performance. Lee *et al.* [8] presented an approach for reducing DRAM miss penalty. In order to hide the latency of flushing dirty DRAM blocks to PCM, a fraction of blocks in DRAM are left empty, ensuring the existence of free space for incoming data. Such under-utilized DRAM space results in more DRAM misses. Another work [9] dynamically resizes the block size in DRAM. Upon a DRAM miss, only part of the missing page is fetched into DRAM. While this technique decreases DRAM miss penalty, it brings extra complexity to each memory access as it requires a two-step process to first check the existence of the page and then identify the existence of the requested block.

To summarize, state-of-the-art optimizations on the hierarchical DRAM-PCM memory organization focus either on reducing the number of PCM writes so as to improve memory endurance, or on reducing the DRAM miss penalty so as to improve memory performance. In contrast, the proposed work aims at reducing the DRAM miss rate which has a direct impact on the performance, endurance, and energy of the hierarchical DRAM-PCM hybrid memory.

III. Technical Motivation

A. Criticality of DRAM Misses

The primary reason for the use of a hybrid main memory is to hide the shortcomings of one device using the strengths of the other. Furthermore, as the main challenges that question the exclusive usage of DRAM are its high standby power and low scalability, the hierarchical organization presented in Fig. 1(b) that uses a relatively small DRAM (about 3% of PCM size) is preferable. This organization is adopted as the baseline memory architecture. In this following parts, the impact of DRAM misses on the performance, energy, and endurance of this hybrid memory will be analyzed.

Equation (1) shows the overall performance of main memory, which is determined by the *average memory hit time* (AMHT), the rate of page faults, and the latency for handling page faults ($T_{\mathrm{PageFault}}$). AMHT is the latency to respond an access which hits either in DRAM or in PCM. In a hierarchical hybrid main memory, since all the memory accesses are directed to DRAM, and PCM is accessed only upon DRAM misses, AMHT is determined by DRAM access latency (T_{DRAM}), DRAM miss rate, and PCM read latency (T_{PCMread})

$$\mathrm{Performance} = \mathrm{AMHT} + \mathrm{Rate}_{\mathrm{PageFault}} \times T_{\mathrm{PageFault}} \quad (1)$$

$$\mathrm{AMHT} = T_{\mathrm{DRAM}} + \mathrm{Rate}_{\mathrm{DRAMmiss}} \times T_{\mathrm{PCMread}}. \quad (2)$$

Equation (2) confirms that the DRAM miss rate has a direct impact on AMHT. Memory accesses that missed in the DRAM are at least 100% slower than accesses that hit in DRAM. Note that (2) assumes that if a memory access misses in DRAM but hits in PCM, the missing content is fetched from the underlying PCM and sent to the CPU directly. This *critical block first* [27] policy effectively reduces DRAM miss penalty since DRAM misses do not need to wait for the page migration between DRAM and PCM, which consists of selecting an eviction target in DRAM, writing it to PCM if it is dirty, and bringing the missing page to DRAM.

Second, the total energy consumed by memory accesses is the sum of the energy consumed by DRAM reads, DRAM writes, PCM reads, and PCM writes. In (3), E_{DRAMread} is the energy of one DRAM read, while N_{DRAMread} is the total number of DRAM reads. Definitions of the other variables are analogous

$$\begin{aligned}\mathrm{Energy} = {} & E_{\mathrm{DRAMread}} \times N_{\mathrm{DRAMread}} \\ & + E_{\mathrm{DRAMwrite}} \times N_{\mathrm{DRAMwrite}} \\ & + E_{\mathrm{PCMread}} \times N_{\mathrm{PCMread}} \\ & + E_{\mathrm{PCMwrite}} \times N_{\mathrm{PCMwrite}}. \end{aligned} \quad (3)$$

The energy consumption is also largely influenced by the number of DRAM misses. Specifically, a memory access hitting in DRAM has the lowest energy cost, since it only causes the read/write of a single DRAM block whose size equals the block size of the LLC. An access missing in DRAM but hitting in PCM, however, will trigger the read of an entire page from PCM and the write of it to DRAM. Additionally, if that miss results in a dirty eviction, it triggers the read of an entire page in DRAM and the write of it to PCM, while PCM writes are considerably long and costly according to Table I. Similarly, page faults also come with the cost of writing an entire page to DRAM and perhaps the eviction of a dirty page to PCM.

Third, in a hierarchical hybrid main memory, PCM writes occur when DRAM misses trigger eviction. The write endurance of a DRAM-PCM hybrid memory is limited by the maximum number of writes to a PCM cell. Assume that the entire PCM is organized in n pages, endurance is constrained by the PCM page with the maximum write count

$$\text{Endurance} = \max_{i=1}^{n} \text{PCMwrite}_i. \tag{4}$$

B. Opportunities to Reduce DRAM Misses

DRAM misses are mainly due to two reasons. First, the DRAM is not large enough to hold the entire working set of the program, thus resulting in capacity misses. Second, multiple pages with overlapping access intervals are mapped to the same set in the DRAM, thus engendering *conflict misses* [28]. While standard solutions are able to reduce capacity misses by enlarging DRAM size and reduce conflict misses by increasing the way-associativity of the DRAM cache, these solutions bring side effects of larger DRAM footprint, increased access latency and energy, and more complicated tag matching process. In comparison, we exploit the interaction between DRAM and PCM, and develop a proactive strategy to allocate pages considering both program segments and conflict misses in the DRAM.

To reduce *capacity* misses, the intuitive idea is to reduce the number of pages competing for the DRAM space. Our observation is that not every page is necessary to be placed in and accessed through DRAM. Instead, as a program has different segments with distinct access patterns and write characteristics, it is preferable to adopt different strategies for their placement. The text segment stores the code and is usually read-only. As shown in Table I, the latency and energy spent in reading PCM are about the same as reading DRAM. It is therefore preferable to place text pages directly in PCM, and bypass DRAM when accessing those pages. This allows the DRAM space to be more efficiently used to store writable data of other segments, including the *data* segment that holds statically and dynamically allocated global variables and arrays, and *stack* segment that stores local variables and function parameters. Accesses to the data and stack segments display high spatial and temporal localities and the set associative structure of DRAM cache is suitable to hold those pages.

To reduce *conflict* misses, pages with overlapping access intervals should be mapped to different DRAM sets.

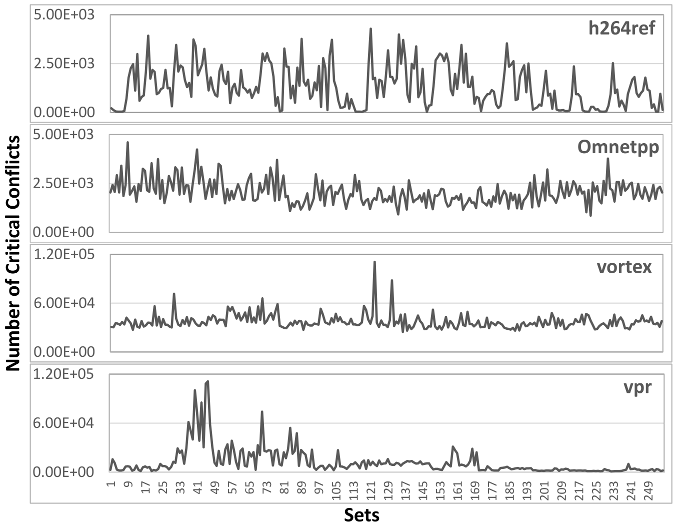

Fig. 2. Distribution of critical conflict misses across DRAM sets. DRAM is 4 MB with 256 sets, while PCM is 128 MB.

Unfortunately, this is not the default case. In fact, many programs have nonuniform memory access patterns, resulting in high variations in the number of conflict misses across various DRAM sets. Fig. 2 shows the number of *critical conflict misses* across various DRAM sets in a hierarchical DRAM-PCM architecture. The memory traces of these SPEC benchmarks are extracted using Pin [29], with an LLC of 1 MB cache size, 128 MB block size, and 4-way associativity modeled during trace extraction. Here, a DRAM miss is classified as a "critical conflict miss" if it hits in PCM and causes a writeback to PCM. As can be seen, in all four benchmarks, the critical DRAM misses are clustered within a few DRAM sets. If those conflicting pages can be more evenly spread across DRAM sets, the number of conflict misses and the resultant PCM writes can be reduced. To achieve this goal, we propose to exploit the flexibility of virtual-to-physical page mapping. Since DRAM is used as a cache for PCM, the exact DRAM set that a page is mapped to is fully determined by its page frame number. Therefore, we propose to leverage DRAM conflict information during page allocation, so as to place a recently accessed page in a less-conflicting DRAM set.

IV. Proposed DRAM-PCM Hybrid Main Memory

A. Segment-Aware Memory Access

The traditional hierarchical DRAM-PCM hybrid memory employs a uniform strategy [11] to handle all page faults and all the memory accesses coming from the LLC. As shown in Fig. 1(b), an access is always directed to DRAM first, and PCM is accessed only upon a DRAM miss. If an access missed in the main memory (i.e., a page fault), the faulting page is always brought into DRAM directly. Pages are written back to PCM only upon DRAM replacements, and only if the replaced page is not in PCM or is dirty.

In contrast, the proposed memory organization adopts different access strategies for different memory segments. The goal is to efficiently utilize the DRAM space for write-intensive

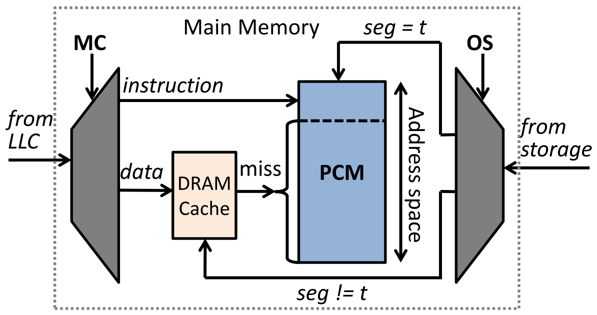

Fig. 3. Proposed DRAM-PCM hybrid architecture and the segment-aware policy for handling page faults and accesses from LLC.

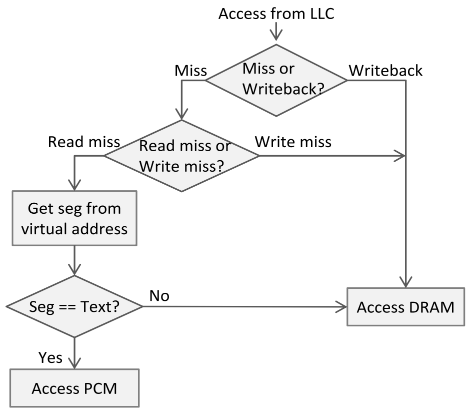

Fig. 4. Access target selection upon an LLC triggered memory access.

data so as to reduce capacity misses. This is achieved by two complementary optimizations, as shown in Fig. 3.

1) An access from LLC is directed to DRAM only if it is not to the text segment, while accesses to the text segment are routed directly to PCM.
2) Upon a page fault, the OS allocates the page and sends its segment information to the MC, which places text pages in PCM and data/stack pages in DRAM.

Implementation of the proposed segment-aware access strategy requires the identification of segment information upon page faults and upon memory accesses from LLC. The former case is straightforward since the OS, who is in charge of handling page faults, has segment information. In the latter case, the target device to access can be conditionally selected based on the type of LLC accesses, using the procedure shown in Fig. 4. Specifically, memory write accesses are caused by LLC writebacks,[1] upon evicting dirty blocks in the LLC. These accesses are directly routed to DRAM since LLC writebacks do not target any text page. On the other hand, memory read accesses are triggered by misses in the LLC. Since cache misses are caused by CPU accesses which use virtual addresses, segment information can be determined accordingly. Furthermore, since text pages are read-only, LLC write misses can be routed to DRAM directly. Therefore, only upon a read miss in the LLC, its virtual address is used to identify segment information, and the access is routed to PCM only if it is to the text segment.

Since different access strategies are adopted for different memory segments, the AMHT should be calculated using the following equation instead of (2):

$$\begin{aligned}\text{AMHT} = {} & \text{Rate}_{\text{text}} \times T_{\text{PCMread}} + (1 - \text{Rate}_{\text{text}}) \\ & \times (T_{\text{DRAM}} + \text{Rate}_{\text{DRAMmiss}} \times T_{\text{PCMread}}). \end{aligned} \quad (5)$$

Here, $\text{Rate}_{\text{text}}$ represents the ratio of text accesses over all memory accesses, while $\text{Rate}_{\text{DRAMmiss}}$ represents the ratio of DRAM misses over all DRAM accesses.

B. Conflict-Aware Page Allocation

To reduce the number of *conflict misses* in DRAM, this section presents an algorithm that proactively allocates a faulting page to a less-conflicting DRAM set. This is achieved

[1]The LLC is assumed to adopt a writeback policy, which is the case in most computer systems.

from two complementary aspects. First, the MC is extended to identify conflict misses in DRAM and record them efficiently. Second, the page allocation algorithm in OS is enhanced to compare DRAM sets and select one with fewer conflicts to hold a faulting page.

1) Recording Conflicting Misses: If an access misses in DRAM but hits in PCM, the miss could be eliminated if the DRAM size/associativity becomes larger. The proposed algorithm considers this case as a *conflict miss*. To keep track of the number of conflict misses across DRAM sets, a hardware counter could be dedicated to each set. However, as indicated in Fig. 2, the number of conflict misses could be very large. Using large counters to record these values, although accurate, would impose sizable storage overhead. Furthermore, since the more recent conflict behavior is more meaningful, it is also unnecessary to use large counters to record all conflict misses. Considering these facts, the proposed technique uses a small N-bit saturating counter per DRAM set. The value of N should be large enough to record recent conflicts, and small enough for counters to be size and power efficient. A detailed study on the size of N will be shown in Section V-B3.

As the MC manages the communication between DRAM and PCM, it is extended to increment these counters upon DRAM misses. Depending on the relative importance of memory performance verse endurance, different priorities can be given to different conflict misses. If performance is the primary concern, all conflict misses are treated equally. On the contrary, if endurance is more important, the counter is incremented by 2 upon a *critical conflict miss* that causes a writeback, and by 1 in other cases. Overall, the MC can adaptively balance performance and endurance, by incrementing counters with different values at various stages of PCM lifetime.

2) Proposed Page Allocation Algorithm: Upon a page fault, if the faulting page is not in the text segment, the proposed algorithm will allocate it to a physical page frame that maps to a less-conflicting DRAM set. To accelerate this process, page frames are organized in multiple lists, as shown in Fig. 5. One list is dedicated to each DRAM set, and all the page frames that map to the same set are placed in the same list.

While the least-conflicting DRAM set can be identified by comparing all the conflict counters, this approach not only slows down page allocation but is also unnecessary since only a small N-bit counter is maintained for each DRAM set.

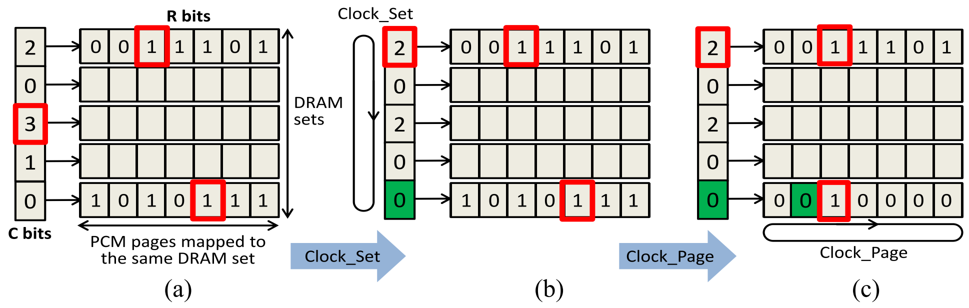

Fig. 5. Two-level hierarchical clock algorithm. Bold frames denote clock hands, while shaded boxes denote the selected set and/or page. (a) Counter values and pointer positions at the time of page fault. Status after applying (b) Clock_Set, which operates on conflicts **(C)** bits of different DRAM sets. Set 5 is selected and (c) Clock_Page, which operates on reference Ⓡ bits of different page frames within the same set. Page 2 in set 5 is selected.

Instead, the proposed approach takes advantage of the classical clock algorithm [30] and applies it hierarchically to first find a *less-recently-conflicting* set and then a *less-recently-accessed* page in that set. The algorithm maintains a set of *clock hands*, one at the higher level and one in each DRAM set, shown as red frames in Fig. 5.

Upon a page fault, the algorithm sequentially scans the conflict counters one by one. If a counter value is nonzero, it is decremented by 1. Otherwise if it is zero, the corresponding DRAM set is selected and the set-level scanning process terminates. Fig. 5 presents a concrete example illustrating this process. For demonstration purposes, 2-bit saturating counters, denoted as C bits, are used to record the recent conflicts in each set. Fig. 5(a) shows counter values and pointer positions at the moment of the page fault, when the *set clock hand* points to the third set (from top to bottom). Since the counter value is nonzero, the algorithm decrements it by one and moves the hand to the next counter. This process repeats until the clock hand reaches the fifth set whose counter value is zero. This set is selected for holding the faulting page. Counter values and pointer positions at the end of this process are shown in Fig. 5(b). Clearly, the selected set is a *less-recently-conflicting* set since it has fewer conflict misses than all the other sets that have been scanned.

An analogous approach can be used to find a *less-recently-accessed* page frame in the selected *less-recently-conflicting* set, based on the reference (R) bits of pages. Specifically, if the selected set is short of free page frames, the Clock_Page process is invoked to scan the R bits, reset nonzero R bits, and select the first page frame whose R bit is zero. This process is shown in Fig. 5(c) where R bits of the fifth, sixth, seventh, and first page frames in the last set have been reset, while the second page frame is selected to hold the faulting page.

This conflict-aware page allocation algorithm is applied to data/stack pages that are to be placed in DRAM. Text pages, on the other hand, are excluded from being placed in DRAM. It is unnecessary to perform conflict-aware allocation to them since they do not cause any DRAM conflict miss. Instead, they will be allocated in a round robin manner. This in turn guarantees that the remaining page frames used to hold data/stack pages are evenly distributed over different DRAM sets.

Overall, the proposed page allocation algorithm is shown in Algorithm 1. It takes three sets of inputs: segment information of the faulting page, conflict counters of different DRAM sets, and reference bits of different page frames. Page allocation consists of three steps. First, the set for holding the faulting page is selected based on segment information (lines 1–6). A nontext page is placed in a *less-recently-conflicting* set, selected using the **Clock_Set** algorithm in the way described above, while a text page is placed in the set pointed by TSetPtr, a global pointer that keeps track of the set for holding the next faulting text page. In the second step, the algorithm selects a page frame from the selected set (lines 7–11). If the set is short of free frames, the **Clock_Page** algorithm is invoked to select a *less-recently-accessed* page frame. On the contrary, if there are free frames available, the first element in the free list is used. Finally, the selected page frame is used to hold the faulting page (line 12).

Algorithm 1 Segment and Conflict-Aware Page Allocation

```
Input:
      seg – segment of the faulting page;
      C[] – conflict counters of different DRAM sets;
      R[] – reference bits of different page frames.
 1: if seg != 'text' then
 2:     selected set ← Clock_Set(C[]).
 3: else
 4:     selected set ← TSetPtr;
 5:     TSetPtr ← (TSetPtr+1)%TotalSets.
 6: end if
 7: if No element in Free[selected set] then
 8:     selected frame ← Clock_Page(R[], selected set).
 9: else
10:     selected frame ← 1st element in Free[selected set].
11: end if
12: Allocate selected frame to faulting page.
```

3) Complexity and Overhead Analysis: Compared to the conventional clock algorithm, the proposed two-level hierarchical page allocation is less complex. Assume n is the total number of in-use pages and m is the total number of DRAM sets. The conventional clock algorithm keeps all the in-use pages in a single list and scans them one by one to find a *less-recently-accessed* page for eviction. Therefore, its worst-case complexity is $O(n)$. In comparison, the proposed algorithm divides the in-use pages into m lists each containing n/m pages. To find a *less-recently-accessed* page, it takes $O(m)$ time to select a list and then $O(n/m)$ time to scan the pages in the selected list. Therefore, the complexity of the proposed two-level process is $O(\max(m, n/m))$.

The proposed page allocation algorithm does not cause any latency overhead because page allocation is performed when the system is waiting for incoming data from secondary storage, and its latency can be hidden completely. Yet it imposes storage overhead since an N-bit saturating counter is required per DRAM set. However, such overhead is small since counters of 2–3 bits are sufficient. As a specific example, for a 32 MB, 4-way associative DRAM, 2048 counters are needed. The entire storage overhead is $3 \times 2048 = 6144$ bits out of 32 MB, equal to 0.002%. These counters are updated upon DRAM misses by the MC when it is waiting for incoming data from PCM.

C. Conflict-Aware Page Remap

Since the page allocation algorithm exploits the recent history of conflict misses to reduce future conflict misses, its effectiveness hinges on the existence of a repetitive pattern of conflict misses. While this is true in most cases, sometimes a DRAM set with few conflict misses in the past may become a highly conflicting set after mapping a new page to it. To reduce conflicts between pages already mapped to the same DRAM set, we propose a further enhancement, i.e., an algorithm that remaps pages from a highly-conflicting DRAM set to a less-conflicting set.

While the idea of remap is straightforward, designing a high-qualify remap technique effective for applications with distinct memory access patterns is challenging. Specifically, as remap requires the swap and hence the write of two entire pages in PCM, it degrades memory endurance and engenders high energy overhead if it is performed aggressively. In light of these side effects, our strategy is to perform remap conservatively without triggering extra PCM writes. In particular, the best time to remap a page is upon writing it to PCM, that is, when the (dirty) page is evicted from DRAM. Since the page has to be written to PCM in any case, remapping it to a free page frame will not induce any extra PCM write on top of the write caused by eviction.

1) Proposed Page Remap Algorithm: The proposed algorithm triggers remap only when the following three conditions are satisfied.

1) Upon replacing a dirty page in DRAM and writing it to PCM.
2) When the current PCM page frame already sustains a large number of writes.
3) The new set should have free page frames to eliminate the need for page swapping.

The second condition detects the existence of conflicting pages in a DRAM set: as conflicting pages evict each other and trigger writebacks to the PCM, the write count of a PCM page frame directly reflects the number of critical conflicts suffered by that page. The first and the third conditions together ensure that remap does not impose any extra PCM writes. Note that in most systems, the third condition is always satisfied since the OS needs to maintain a set of free page frames to shorten the latency of serving incoming page faults. The proposed technique additionally guarantees that these free page frames are evenly distributed across DRAM sets.

Algorithm 2 Conflict-Aware Page Remapping

```
Inputs:
    evicted page;
    WC[] – write counts of PCM pages;
    C[] – conflict counters of different DRAM sets;
    R[] – reference bits of different page frames.
1: if evicted page is dirty then
2:     if WC[evicted page] $\geq$ Remap_Threshold then
3:         new page $\leftarrow$ Page Allocation ('non-text', C[],R[]);
4:         WC[new page] $\leftarrow$ 1;
5:     else
6:         WC[evicted page]++;
7:     end if
8: end if
```

Algorithm 3 Adapting *Remap_Threshold*

```
Inputs:
    W – Number of writes since last Remap_Threshold adaptation;
    R– Number of remaps since last Remap_Threshold adaptation;
    Nset – Number of sets in DRAM;
1: Remap_Threshold $\leftarrow$ Initial value;
2: if (R>Nset) or (W>Nset$\times$Remap_Threshold$\times$2) then
3:     Remap_Threshold $\leftarrow$ Remap_Threshold + Step;
4:     W $\leftarrow$ 0;
5:     R $\leftarrow$ 0;
6: end if
```

Pseudo code of the proposed conflict-aware page remapping algorithm is shown in Algorithm 2. Upon a dirty eviction from DRAM, the MC checks the write count of that page frame in the PCM (lines 1 and 2). A Remap_Threshold is used to select remapping targets. This threshold is the upper bound of critical conflicts that a PCM page frame is allowed to sustain contiguously. Once the write count of a PCM page reaches to Remap_Threshold, the MC invokes page allocation (Algorithm 1) to remap the highly-conflicting virtual page to a free frame in less-conflicting DRAM set (line 3). The write count of the new physical page frame is set to 1 (line 4).

2) Threshold Adaptation: Clearly, the number of remaps is determined by the value of the Remap_Threshold. However, a single and fixed Remap_Threshold is not able to meet the requirements of different applications which produce highly diverse memory access patterns and write characteristics. Instead, the proposed scheme uses a low initial value for the Remap_Threshold, and increases it based on application needs. To trigger threshold adaptation, two values are continuously monitored: the number of remaps and the number of writes to PCM, both since the last adaptation of Remap_Threshold. If a considerable number of remaps has been performed since the last adaptation, the Remap_Threshold is increased to lower the remap frequency. Furthermore, a considerable number of PCM writes accompanied by relatively few remaps implies that writes are distributed across different sets more-or-less evenly. In this case, remap is unnecessary, and the Remap_Threshold is increased to eliminate potential unproductive remaps.

Algorithm 3 shows the exact conditions and procedure for adapting the Remap_Threshold. The threshold is increased if the number of remaps is more than the number of sets in DRAM, implying that on average one remap is performed per set, or if the number of PCM writes is twice as much as the number of sets multiplied by the Remap_Threshold. Note that W/R is always larger than Remap_Threshold. Therefore if ($W > Nset \times$ Remap_Threshold $\times$ 2) but ($R \leq Nset$), it implies that more than 50% of PCM writes are performed without triggering any remap. In other words, PCM writes are more-or-less balanced. If either condition is true (line 2), the Remap_Threshold is increased linearly (line 3), and *Step* equals half of the *Initial value* of Remap_Threshold.

3) Complexity and Overhead Analysis: Since the remap algorithm calls the proposed two-level hierarchical clock algorithm, it has the same complexity as page allocation. Furthermore, remap is not on the critical path either, since as mentioned in Section III-A, a DRAM miss does not need to wait for the page migration between DRAM and PCM. To avoid adding extra storage overhead, the proposed technique takes advantage of write counters used by many wear-leveling techniques [31]. As the upper bound value of these counters equals the maximum value of Remap_Threshold, the size of each counter is $\log_2\lceil$Remap_Threshold$\rceil$. Even with a counter size of 4-byte which is large enough to hold the value of the wear-out limit (10^9), the storage overhead is 0.1% assuming a typical page size of 4 KB.

D. Write Count-Aware Page Replacement

The replacement policy used in DRAM impacts both the performance and endurance of the hybrid main memory. While many caches use *least-recently-used* (LRU) or pseudo LRU policies to reduce miss rates, some hybrid structures prefer to replace clean pages in DRAM to reduce the number of PCM writes. As an example, the *N-chance* algorithm [10], when applied to a W-way associative cache, checks N blocks ($1 \leq N \leq W$) one by one starting from the LRU position, and selects the first clean block as the eviction target. If none of the N blocks are clean, the LRU block will be evicted.

While the N-chance policy is able to reduce the number of PCM writes in most cases, it does not guarantee endurance improvement. This is because PCM endurance is limited by the maximum number of writes to a page frame. However, when the N-chance algorithm replaces a dirty page due to the lack of a clean page, the write count of that dirty page is not taken into consideration. Another drawback is that as N becomes larger, the probability of replacing a more recently used block increases, thus resulting in more DRAM misses.

Being aware of these limitations, we propose a write count-aware page replacement policy that sacrifices DRAM performance only for reducing PCM writes that degrade endurance. This can be illustrated through a example of two pages A and B in the same DRAM set with the following conditions.

1) B is accessed more recently than A.
2) A is dirty while B is clean.
3) Regarding the write counts of the corresponding page frames in PCM, A has fewer writes than B.

In this example, LRU selects A for eviction while N-chance selects B. From the performance perceptive, evicting A, the less recently accessed page, is the better choice. From the endurance perspective, however, there is no difference between the two pages since evicting either A or B does not increase the maximum writes to a PCM page. In this case, the proposed replacement policy makes a performance-beneficial decision by evicting A. On the other hand, if the third condition flips (i.e., A has more writes than B), the proposed policy makes a endurance-beneficial decision by evicting B.

To obtain age information of each PCM page, the proposed DRAM replacement policy leverages the write counts used by many page-level wear-leveling techniques of PCM [17]. A higher priority is given to either a clean page or a dirty page whose corresponding PCM write counter has a smaller value. The complete replacement algorithm is shown in Algorithm 4. Same as LRU and N-chance, this algorithm assumes the existence of an LRUorder, a list wherein pages are sorted based on their access recency. Similar to the N-chance algorithm, this algorithm has a search range of N pages, starting from the LRU position. It iteratively compares page p against page $p+1$, and selects page p as the eviction target if it is clean or if its write count is smaller than that of page $p+1$. At the end if none of the N pages satisfy these two conditions, the page at the LRU position is selected for eviction. Overall, a small value of N makes the algorithm performance friendly, while a large value of N makes it endurance friendly.

Algorithm 4 Write Count-Aware Page Replacement in DRAM

```
Input:
    WC[] – write counts of PCM pages;
    LRUorder[] – LRU order of pages in a DRAM set.
Output:
    Eviction_Target.
 1: $p \leftarrow 0$;
 2: Eviction_Target ← LRUorder[0];
 3: Next_Page ← LRUorder[1];
 4: while ($p < N$) and (Eviction_Target is dirty) and
    (WC[Eviction_Target] > WC[Next_Page]) do
 5:     $p \leftarrow p + 1$;
 6:     Eviction_Target ← Next_Page;
 7:     Next_Page ← LRUorder[$p + 1$];
 8: end while
 9: if ($p = N$) then
10:     Eviction_Target ← LRUorder[0];
11: end if
12: return Eviction_Target.
```

V. Experimental Evaluation

In this section, the proposed segment and conflict aware hybrid DRAM-PCM main memory is evaluated from two aspects. First, a self analysis is conducted to show the impact of each component of the proposed hybrid memory. Various sets of configurations are examined to identify the best one. Second, the proposed hybrid memory is compared to other representative hierarchical DRAM-PCM structures in terms of their performance, energy, and lifetime.

A. Experiment Setup

We have developed a memory management system to simulate the proposed DRAM-PCM hybrid architecture and the proposed page allocation, remap, and replacement algorithms.

TABLE II
MEMORY SYSTEM CONFIGURATION

Component	Configuration
Last Level Cache	Size: 1MB block size: 128B Associativity: 4
DRAM	Size: 32MB Cell type: LP-DRAM Number of banks: 8 Read/Write Ports per bank: 1 Associativity: 4
PCM	Size: 1GB Bank Organization: 32 x 64 Mat Organization: 2 x 2

TABLE III
ATTRIBUTES OF 1 GB PCM AND 32 MB DRAM REPORTED BY NVSIM [32] AND CACTI [33] ASSUMING 32 nm TECHNOLOGY

Attributes	PCM	DRAM
Area (mm^2)	120.45	316.13
Read Latency (ns)	62.57	15.83
Write Latency (ns)	322.96	15.83
Read Energy (nJ)	1.71	99.39
Write Energy (nJ)	81.14	99.39
Leakage Power (W)	5.22	1.41

Sizes of PCM and DRAM are set to 1 GB and 32 MB (DRAM is 3% of PCM), respectively. The page size is 4 KB.

NVsim [32] and CACTI [33] are used to estimate the latency and energy parameters of PCM and DRAM, respectively. Detailed configuration parameters are listed in Table II. These parameters are selected to optimize the access latency of the device. Table III shows the latency and energy parameters reported by NVsim and CACTI. The read latency of the PCM is about four times longer than DRAM read latency, which seems to contradict with the discussion in Section II-A. This is due to the fact that the PCM array is 32 times larger than the DRAM array, thus it requires a larger decoder with longer delay which dominates the read latency. For the same reason, the PCM has larger leakage power than the DRAM.

The evaluation set includes 19 benchmarks from the SPEC 2000 and 2006 suites. Each benchmark is executed on a 2.4GHz 8-core server, while Pin [29] is used to extract memory access traces, including the address, the operation (read or write), and the segment. An LLC whose configuration is given in Table II is modeled in Pin when extracting these traces. To simulate a multiprocess environment, eight benchmarks are executed in parallel. Five different groups are used to represent a diverse set of workload combinations. Table IV shows the benchmarks in each group.

The hierarchical DRAM-PCM structure proposed in [11] is considered as the baseline architecture. Its organization and access policy is shown in Fig. 1(b). Table V shows the rate of DRAM misses (out of total memory accesses), the rate of PCM writes (out of total memory writes), the AMHT computed with (2), the maximum writes on a PCM page, and the total energy consumption of the baseline memory architecture.

B. Self-Analysis

The first set of experiments examines the impact of the different techniques proposed in Section IV, namely the segment-aware access policy, the segment- and conflict-aware page allocation, and the conflict-aware remap algorithm. In all these studies, LRU is the replacement strategy used in DRAM. The impact of the proposed write count-aware replacement algorithm will be studied in the next set of experiments.

1) Reduction in DRAM Misses: Overall, the proposed scheme is quite effective in reducing capacity misses and balancing conflict misses across different DRAM sets. Fig. 6 shows the number of *critical conflicts* across various DRAM sets for all the five groups. It can be seen that by allocating and remapping pages to less-conflicting sets, the proposed scheme effectively reduces the standard deviation of misses in different DRAM sets by 64.78%.

TABLE IV
GROUPS OF BENCHMARKS

Group Name	benchmarks
Group1	gromacs, h264ref, hmmer, leslie3d, omnetpp, sjeng, tonto, zeusmp
Group2	ammp, gromacs, leslie3d, mgrid, milc, sjeng, tonto, wupwise
Group3	bwaves, gzip, leslie3d, milc, sjeng, vortex, wupwise, zeusmp
Group4	facerec, gzip, h264ref, mgrid, omnetpp, tonto, vpr, wupwise
Group5	facerec, gamess, gcc, gzip, h264ref, vortex, vpr, zeusmp

TABLE V
PROPERTIES OF THE BASELINE HYBRID MEMORY ARCHITECTURE

	DRAM miss rate	PCM write rate	AMHT (ns)	Max write count	Energy (J)
Group1	2.72%	5.84%	17.53	7,578	1000.85
Group2	3.16%	7.79%	17.81	11,424	1252.25
Group3	1.14%	2.11%	16.54	1,634	764.33
Group4	1.75%	4.00%	16.92	2,866	749.16
Group5	3.15%	7.71%	17.80	8,145	1011.09
Average	2.38%	5.49%	17.32	6,329	955.53

TABLE VI
THREE IMPLEMENTED SCHEMES FOR SELF EXAMINATION

Schemes	SA	CA	RM
Segment-Aware Access Policy	✓	✓	✓
Conflict-Aware Page Allocation		✓	✓
Conflict-Aware Remap Algorithm			✓

TABLE VII
REDUCTION IN DRAM MISSES OVER THE BASELINE ARCHITECTURE

Benchmark	SA + CA + RM = Total
Group1	0.18% + 3.16% + 3.92% = 7.26%
Group2	0.06% + 6.28% + 2.09% = 8.43%
Group3	1.60% + 2.56% + 1.25% = 5.41%
Group4	0.76% + 4.05% + 1.83% = 6.64%
Group5	0.05% + 6.81% + 1.81% = 8.67%
Average	0.53% + 4.57% + 2.18% = 7.28%

TABLE VIII
REDUCTION IN PCM WRITES OVER THE BASELINE ARCHITECTURE

Benchmark	SA + CA + RM = Total
Group1	0.12% + 1.95% + 3.94% = 6.01%
Group2	0.05% + 3.64% + 2.66% = 6.35%
Group3	0.48% + 2.44% + 2.22% = 5.13%
Group4	0.31% + 2.05% + 3.91% = 6.26%
Group5	0.04% + 3.60% + 3.45% = 7.10%
Average	0.20% + 2.74% + 3.23% = 6.17%

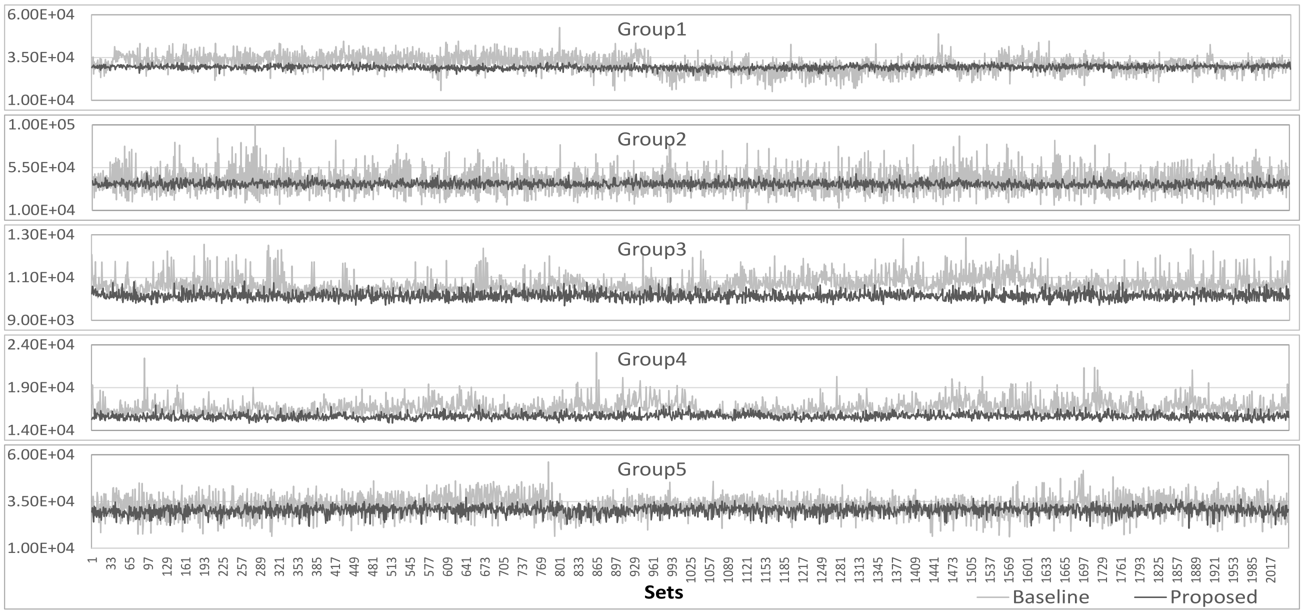

Fig. 6. Effectiveness of proposed technique in balancing conflict misses across different DRAM sets.

2) Contributions of Each Component: To determine the improvement achieved by each of the proposed techniques, they are accumulatively applied one by one. Three configurations denoted as “SA,” “CA,” and “RM” are studied, with the techniques employed in each configuration listed in Table VI. Two-bit conflict counters are used in these studies, while the Remap_Threshold is initialized to 128. Tables VII and VIII, respectively, present the reduction in DRAM misses and PCM writes achieved by each component. The SA columns report the reduction over the baseline architecture, while CA is the reduction on top of SA and RM is the reduction on top of CA. Finally, the “total” columns show the overall reduction in DRAM misses and PCM writes over the baseline architecture when all three components are applied.

The segment-aware access strategy effectively reduces DRAM capacity misses when the code size of the running applications is significant and the number of accesses to the code segments is noticeably high. Among the five groups, *Group3* has the highest number of text accesses and hence displays the highest reductions in the SA column.

A comparison between the columns in Table VII shows that conflict-aware page allocation, on average, is the most effective technique in reducing DRAM misses. It consistently outperforms the segment-aware access strategy. It also outperforms the remap algorithm for groups 2–5, indicating that after proactively allocating pages based on conflict history, the standard deviation of misses across different DRAM sets is quite low for those four groups. On the other hand, for *Group1*, more potential is left for the remap algorithm to detect and resolve the highly-conflicting sets.

Similarly, a comparison between the columns in Table VIII shows that the remap algorithm, on average, is the most effective technique in reducing PCM writes. This is because it directly monitors the writebacks to PCM to identify conflicts. Among the three schemes, RM is the only one that delivers a higher reduction in PCM writes (3.23% in Table VII) than in DRAM misses (2.18% in Table VIII).

3) Impact of Conflict Counter Size: As mentioned in Section IV-B1, using conflict counters of larger sizes may improve the accuracy of selecting a less-conflicting set, but imposes more storage overhead and causes the **Clock_Set** algorithm to perform more scan operations. Fig. 7 shows the changes in DRAM miss rate, PCM write count, and the number of scans as the counter size increases from 1 to 5 bits. The miss rate and write count are normalized to the corresponding values in the baseline architecture, while the number of scan operations is normalized to the maximum value among all the configurations tested for each benchmark. In each graph, smaller values are preferred.

As Fig. 7 shows, larger counter sizes do not always deliver fewer DRAM misses and PCM writes. In other words, although large counters are able to record longer conflict history, information of older conflicts may not be helpful in selecting a set with fewer conflicts recently. Fig. 7 shows that a counter size of 2–3 bits is best for performance. However, as the scan overhead increases by 57% when the counter size increases from 2 to 3 bits, 2-bit is the recommended counter size and is used in the rest of studies.

4) Impact on Page Faults: As (1) shows, the overall memory performance is determined by both AMHT and the number of page faults. While the proposed techniques effectively reduce the number of DRAM misses and hence improve AMHT, it is necessary to ensure that this benefit does not come at a price of causing more page faults, which are orders of magnitude more costly than memory accesses. To clarify this point, Table X presents the rates of page faults in both the baseline and the proposed work. As can be seen, for four out of the five tested groups, the proposed scheme does not cause any change to the page fault rate. The only observable difference is in Group2 whose page fault rate is increased by 10^{-5}%, which is negligible.

TABLE IX
ATTRIBUTES OF APPROACHES IMPLEMENTED FOR COMPARISON

Schemes	**Memory access policy**	**Page allocation algorithm**	**Remap algorithm**	**DRAM replacement policy**	**Mandatory empty block per DRAM set**
Baseline [11]	Access DRAM first Access PCM on DRAM miss	Conventional clock [30]	No remap	LRU	No
N-chance [10]	Access DRAM first Access PCM on DRAM miss	Conventional clock [30]	No remap	3-chance clean first	No
Reduced miss penalty [8]	Access DRAM first Access PCM on DRAM miss	Conventional clock [30]	No remap	LRU	Yes
Proposed_LRU	Proposed segment aware	Proposed conflict aware (Clock_Set + Clock_Page)	Conflict aware remap	LRU	No
Proposed_WA	Proposed segment aware	Proposed conflict aware (Clock_Set + Clock_Page)	Conflict aware remap	Proposed write count aware	No
Proposed_3C	Proposed segment aware	Proposed conflict aware (Clock_Set + Clock_Page)	Conflict aware remap	3-chance clean first	No

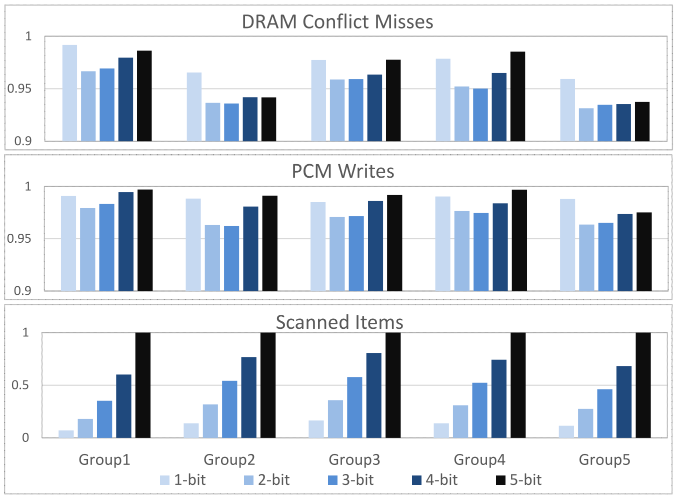

Fig. 7. Impact of conflict counter size on DRAM misses, PCM writes, and the number of scan operations.

TABLE X
COMPARISON OF PAGE FAULT RATES

Benchmark	Baseline	Proposed
Group1	0.00308%	0.00308%
Group2	0.01654%	0.01655%
Group3	0.00340%	0.00340%
Group4	0.00405%	0.00405%
Group5	0.00394%	0.00394%
Average	0.00602%	0.00602%

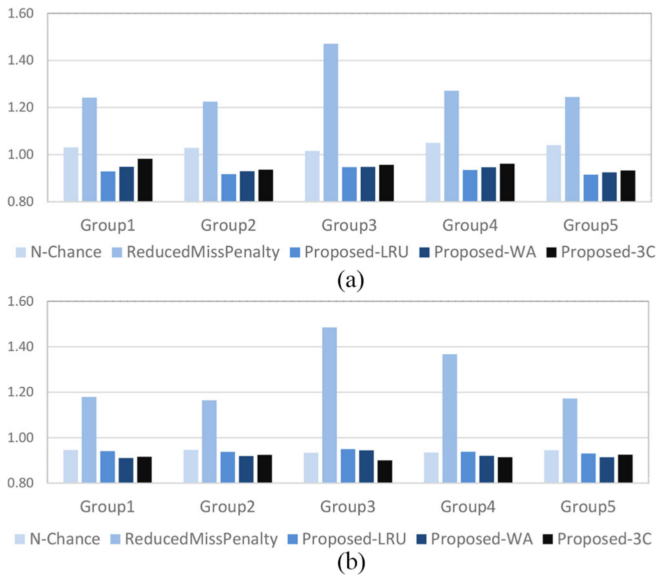

Fig. 8. Comparison of five schemes. All values normalized to baseline. Normalized (a) DRAM conflict misses and (b) PCM writes.

C. Comparison With the State-of-the-Art Techniques

1) Methodology: The second set of experiments compares the proposed hybrid memory with other representative schemes that aim to improve the baseline architecture [11]. In addition to the baseline, five different approaches are implemented and compared in terms of the total numbers of DRAM misses and PCM writes, AMHT, energy consumption, and lifetime. Details of their attributes are summarized in Table IX. First, AMHT is used as the performance metric. The AMHT of the baseline, *N-chance*, and *Reduced Miss Penalty* is calculated with (2), while the AMHT of the proposed schemes is calculated with (5). In both equations, AMHT depends on DRAM misses rather than PCM writes due to the use of *critical block first* [27]. Second, the dynamic energy consumed by each scheme is calculated using (3). Note that the number of DRAM reads is affected by the access policy, while the numbers of DRAM writes, PCM reads, and PCM writes are affected by the number of DRAM misses. Third, Memory lifetime is constrained by the maximum number of writes on a single PCM page. This value is affected by the DRAM replacement policy and the remap algorithm, if any.

2) Results: Fig. 8 shows the numbers of DRAM misses and PCM writes, while Fig. 9 depicts the results of AMHT, energy, and memory lifetime. All the values are normalized to the corresponding baseline results. Therefore, a value less/greater than 1 indicates an improvement/degradation over the baseline.

The results of *N-chance* show the impact of clean-first replacement algorithm, which aims to reduce the writes to PCM and improve its lifetime by replacing clean pages in DRAM. Fig. 8 shows that on average, *N-chance* reduces PCM writes by 6% but increases DRAM misses by 3% compared to the baseline that adopts LRU. Consequently, it increases AMHT by 0.8% and energy by 0.7%, as shown in Fig. 9. For *Group3*, *N-chance* achieves a slight energy reduction because the extra DRAM misses induced by *N-chance* are fewer than other groups, which can be observed from Fig. 8(a). In terms of memory lifetime, *N-chance* is able to reduce the maximum write count on a PCM page by 32% on average.

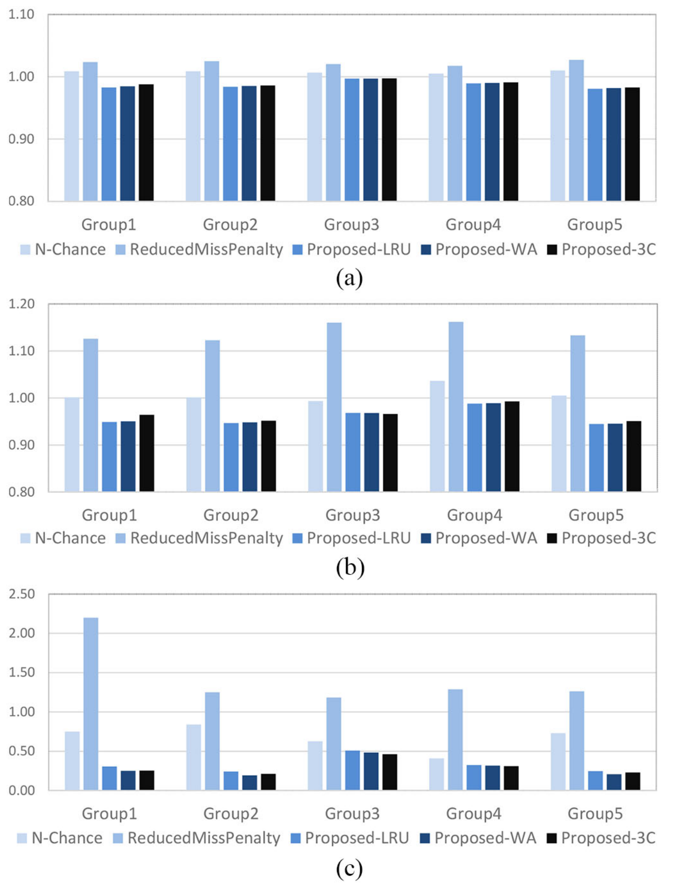

Fig. 9. Comparison of five schemes. All values normalized to baseline. Normalized (a) AMHT, (b) Energy consumption, and (c) maximum write count on a page.

The results of *Reduced Miss Penalty* show the importance of DRAM misses. As mentioned before, this scheme reduces the miss penalty by maintaining an empty page per DRAM set, thus eliminating the need for eviction upon DRAM misses. However, this scheme reduces the effective size of DRAM by 25% (since the DRAM in experiment is 4-way associative) and hence incurs 29% more DRAM misses and 27% more PCM writes on average, as shown in Fig. 8. More critically, as the *critical block first* technique [27] is adopted by all other schemes, the benefit of *Reduced Miss Penalty*, in hiding the latency of PCM writes on the critical path becomes in-distinctive. Thus, *Reduced Miss Penalty* shows the worst AMHT, energy consumption, and lifetime among all the schemes in Fig. 9.

The *Proposed_LRU* adopts the same DRAM replacement policy as the baseline, but consistently outperforms the baseline in reducing both DRAM misses and PCM writes, as shown in Fig. 8. As a result, it is able to reduce AMHT and energy consumption and prolong PCM lifetime at the same time. Similarly, the *Proposed_3C* adopts the same replacement policy as *N-chance* but consistently outperforms *N-chance*. These results prove the compatibility of the proposed technique with various DRAM replacement policies.

Among the three variations of the proposed technique, on average *Proposed_LRU* is the best one in reducing DRAM misses, followed by *Proposed_WA*. However, *Proposed_3C* achieves the highest reduction in PCM writes, followed by *Proposed_WA*. These results prove the effectiveness of write count aware replacement in trading-off between DRAM misses and PCM writes. Overall, *Proposed_LRU* delivers the shortest AMHT among all the schemes, at a 1.3% average reduction over the baseline, while *Proposed_WA* and *Proposed_3C* reduce the AMHT by 1.2% and 1.1%, respectively. In terms of energy, *Proposed_LRU*, *Proposed_WA*, and *Proposed_3C* consume 4.1%, 3.9%, and 3.4% less energy than the baseline, respectively. In terms of memory lifetime, the three variations of LRU, WA, and 3C reduce the maximum number of writes on a PCM page over the baseline by 67.5%, 72.0%, and 70.5%, respectively. These results confirm the advantage of the proposed *write count aware replacement algorithm* over the *3-chance clean first algorithm*, by considering the write count of an evicted page rather than merely checking whether the page is dirty or clean.

To summarize, the comparison among the six schemes confirms that all the three variations of the proposed technique consistently outperform the other three approaches in reducing DRAM misses and PCM writes, shortening AMHT, saving energy, and prolonging PCM lifetime.

VI. Conclusion

This paper has presented the design of a hybrid DRAM-PCM main memory and demonstrated the criticality of DRAM misses to memory performance, energy consumption, and lifetime. To reduce DRAM misses, this paper explores the interaction between DRAM and PCM: capacity misses in DRAM are reduced by placing read-only pages in PCM and bypassing DRAM upon their accesses; conflict misses in DRAM are reduced by proactively allocating pages to less-conflicting DRAM sets. A conflict-aware remap strategy is proposed to further reduce conflict misses and prolong PCM lifetime without imposing extra PCM writes. A write count-aware replacement policy is also proposed to balance memory performance and lifetime. Intensive experimental studies on SPEC 2000 and 2006 benchmarks show that the proposed techniques are able to deliver 7% reduction in DRAM misses and 6% reduction in PCM writes. As a consequence, they are able to improve AMHT by 1.3%, reduce energy consumption by 4%, and prolong PCM lifetime by 72%. These results confirm that unlike traditional techniques which have to trade-off among performance, energy consumption, and lifetime, the proposed hybrid memory design is able to improve all the three factors at the same time.

References

[1] ITRS, "International technology roadmap for semiconductors," Emerging Research Devices (ERD). [Online]. Available: http://www.itrs2.net/2013-itrs.html

[2] S. Raoux *et al.*, "Phase-change random access memory: A scalable technology," *IBM J. Res. Develop.*, vol. 52, no. 4, pp. 465–479, 2008.

[3] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Austin, TX, USA, 2009, pp. 14–23.

[4] A. N. Udipi, "Rethinking DRAM design and organization for energy-constrained multi-cores," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Saint-Malo, France, 2010, pp. 175–186.

[5] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2009, pp. 664–669.

[6] Y. Park, D.-J. Shin, S. K. Park, and K. H. Park, "Power-aware memory management for hybrid main memory," in *Proc. Int. Conf. Next Gener. Inf. Technol. (ICNIT)*, Hong Kong, 2011, pp. 82–85.

[7] 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 Trans. Comput.*, vol. 63, no. 9, pp. 2187–2200, Sep. 2014.

[8] H. G. Lee, S. Baek, C. Nicopoulos, and J. Kim, "An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems," in *Proc. Int. Conf. Comput. Design (ICCD)*, Amherst, MA, USA, 2011, pp. 381–387.

[9] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lees, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Shenzhen, China, 2013, pp. 424–435.

[10] A. P. Ferreira *et al.*, "Increasing PCM main memory lifetime," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, 2010, pp. 914–919.

[11] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. Int. Symp. Comput. Architect. (ISCA)*, Austin, TX, USA, 2009, pp. 24–33.

[12] H. A. Khouzani, C. Yang, and J. Hu, "Improving performance and lifetime of DRAM-PCM hybrid main memory through a proactive page allocation strategy," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Chiba, Japan, 2015, pp. 508–513.

[13] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2009, pp. 347–357.

[14] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *Proc. Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2009, pp. 2–13.

[15] H. A. Khouzani, Y. Xue, and C. Yang, "Fully exploiting PCM write capacity within near zero cost through segment-based page allocation," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 4, 2016, Art. no. 31.

[16] D. Liu *et al.*, "Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Yokohama, Japan, 2013, pp. 279–284.

[17] C.-H. Chen, P.-C. Hsiu, T.-W. Kuo, C.-L. Yang, and C.-Y. M. Wang, "Age-based PCM wear leveling with nearly zero search cost," in *Proc. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 453–458.

[18] M. K. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2009, pp. 14–23.

[19] M. Zhao, L. Shi, C. Yang, and C. J. Xue, "Leveling to the last mile: Near-zero-cost bit level wear leveling for PCM-based main memory," in *Proc. Int. Conf. Comput. Design (ICCD)*, Seoul, South Korea, 2014, pp. 16–21.

[20] M. Asadinia, M. Arjomand, and H. Sarbazi-Azad, "Prolonging lifetime of PCM-based main memories through on-demand page pairing," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 20, no. 2, 2015, Art. no. 23.

[21] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," in *Proc. Int. Symp. Microarchit. (MICRO)*, Atlanta, GA, USA, 2010, pp. 115–124.

[22] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," in *Proc. Int. Symp. Microarchit. (MICRO)*, 2013, pp. 433–444.

[23] R. Melhem, R. Maddah, and S. Cho, "RDIS: A recursively defined invertible set scheme to tolerate multiple stuck-at faults in resistive memory," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Boston, MA, USA, 2012, pp. 1–12.

[24] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories," in *Proc. Int. Conf. Comput. Design (ICCD)*, Montreal, QC, Canada, 2012, pp. 337–344.

[25] W. Wei, D. Jiang, S. A. Mckee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *Proc. Int. Conf. Parallel Archit. Compilation Tech. (PACT)*, San Francisco, CA, USA, 2015, pp. 163–173.

[26] J. Hu *et al.*, "Low overhead software wear leveling for hybrid PCM + DRAM main memory on embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 4, pp. 654–663, Apr. 2015.

[27] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2012.

[28] M. Dubois, M. Annavaram, and P. Stenström, *Parallel Computer Organization and Design*. Cambridge, U.K.: Cambridge University Press, 2012.

[29] (2012). *Programmable Instrumentation—A Dynamic Binary Instrumentation Tool*. [Online]. Available: http://software.intel.com/en-us/articles/pintool

[30] A. S. Tanenbaum, *Modern Operating Systems*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[31] H. A. Khouzani, Y. Xue, C. Yang, and A. Pandurangi, "Prolonging PCM lifetime through energy-efficient, segment-aware, and wear-resistant page allocation," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, La Jolla, CA, USA, 2014, pp. 327–330.

[32] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[33] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

Hoda Aghaei Khouzani received the B.S. degree from the Iran University of Technology, Tehran, Iran, in 2008, and the M.S. degree from the Sharif University of Technology, Tehran, in 2010, both in computer engineering. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA.

Her current research interests include runtime optimizations of nonvolatile memories on various levels of memory hierarchy.

Fateme S. Hosseini received the B.S. degree from the Iran University of Technology, Tehran, Iran, in 2008, and the M.S degree from the Sharif University of Technology, Tehran, in 2011, both in computer engineering. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA.

Her current research interests include compiler optimizations to enhance system reliability.

Chengmo Yang received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from the University of California at San Diego, San Diego, CA, USA.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA. She has published over 50 technical papers at first-tier conferences and journals. Her current research interests include computer architecture, embedded systems, and design automation, with a particular focus on the improvement of reliability, security, nonvolatility, and energy-efficiency of next generation processor and memory systems.

Dr. Yang was a recipient of the University of Delaware Research Foundation Award and the National Science Foundation CAREER Award.