Page Placement in Hybrid Memory Systems

Luiz Ramos
Rutgers University
Piscataway, NJ, USA
luramos@cs.rutgers.edu

Eugene Gorbatov
Intel Corporation
Hillsboro, OR, USA
eugene.gorbatov@intel.com

Ricardo Bianchini
Rutgers University
Piscataway, NJ, USA
ricardob@cs.rutgers.edu

ABSTRACT

Phase-Change Memory (PCM) technology has received substantial attention recently. Because PCM is byte-addressable and exhibits access times in the nanosecond range, it can be used in main memory designs. In fact, PCM has higher density and lower idle power consumption than DRAM. Unfortunately, PCM is also slower than DRAM and has limited endurance. For these reasons, researchers have proposed memory systems that combine a small amount of DRAM and a large amount of PCM. In this paper, we propose a new hybrid design that features a hardware-driven page placement policy. The policy relies on the memory controller (MC) to monitor access patterns, migrate pages between DRAM and PCM, and translate the memory addresses coming from the cores. Periodically, the operating system updates its page mappings based on the translation information used by the MC. Detailed simulations of 27 workloads show that our system is more robust and exhibits lower energy-delay^2 than state-of-the-art hybrid systems.

Categories and Subject Descriptors

C.5.5 [**Computer system implementation**]: Servers

General Terms

Design, Performance

1. INTRODUCTION

Main memory capacity is becoming a critical issue for many systems. As the number of processor cores in each CPU chip increases, so do the number of concurrent threads, applications, and/or virtual machines that must have their working sets simultaneously in main memory. Unfortunately, current trends suggest that meeting these capacity requirements using DRAM will not be ideal. DRAM exhibits low access times, but consumes significant amounts of energy (idle, refresh, and precharge energies). As a result, the amount of energy consumed by the memory is approaching (and sometimes surpassing) that consumed by the processors in many servers [2, 21].

For these reasons, architects have started to consider Phase-Change Memory (PCM) as a potential replacement for DRAM [20, 28, 37, 38]. PCM is byte-addressable, consumes little idle energy, does not require refreshing or precharging (its contents are persistent), and exhibits access times in the nanosecond range. Furthermore, PCM cells have feature size comparable to DRAM cells, but can store more information in the same physical area. However, PCM's read/write times, read/write energies, and write endurance are worse than those of DRAM. To take advantage of the low latencies of DRAM and the high capacity of PCM, researchers have proposed hybrid memory systems that combine a small amount of DRAM and a large amount of PCM [28, 37, 38]. Unfortunately, those systems exhibit poor behavior for certain workloads, as we shall demonstrate. For example, the system proposed in [28] degrades performance significantly for workloads with poor locality, whereas that from [37] suffers when cache write-backs represent a small fraction of the memory traffic.

In this paper, we propose a new DRAM+PCM memory system design that is robust across a wide range of workloads. The design comprises a sophisticated memory controller (MC) that implements a page placement policy called "Rank-based Page Placement" (RaPP). The policy efficiently ranks pages according to popularity (access frequency) and write intensity, migrating top-ranked pages to DRAM. While monitoring popularity, RaPP penalizes pages that are unlikely to produce benefits if migrated. To improve PCM's endurance, each migration involves two PCM memory frames and one DRAM frame. The MC monitors access patterns and, when necessary, migrates pages. The migrations are not immediately visible by the operating system (OS), as the MC uses its own address translation table. Periodically (or when the table fills up), the OS updates its mapping of virtual pages to physical frames based on the translation table and clears it.

We evaluate our memory system design and RaPP using a detailed simulator that computes the energy, performance, and endurance of workloads running on an 8-core CPU. For comparison with our system, we simulate two state-of-the-art hybrid memory designs [28, 37], as well as a baseline hybrid system without page management (called "unmanaged") and a PCM-only system.

Our results for 27 workloads show that our system consumes roughly the same amount of power on average as its competitors and the baselines, but with significantly better performance. In terms of energy-delay^2, our system is on average 36% better than the PCM-only baseline and 24% better than the unmanaged hybrid system. Compared to the state-of-the-art hybrid systems, our system exhibits at least 13% better energy-delay^2 on average,

ICS'11, May 31–June 4, 2011, Tucson, Arizona, USA.

especially for workloads with large memory footprints. Our system also improves lifetime, as compared to the baselines, but not enough to enable system operation for 5 years assuming current PCM endurance. Nevertheless, PCM endurance is expected to increase by orders of magnitude in the next few years [1, 9]. Until then, our system can be easily combined with previously proposed endurance-improvement techniques [8, 18, 20, 38].

We conclude that existing hybrid memory systems are useful, but they need to be more robust and efficient. Our system is a demonstration that these characteristics can be achieved through a sophisticated MC, and careful page ranking and migration.

2. BACKGROUND

A server's memory system typically comprises a few key elements, namely a memory controller (MC), a few memory channels, and a number of dual-inline memory module (DIMMs). Each DIMM includes memory devices (chips) that contain a memory cell array and peripheral circuitry.

The MC is responsible for handling memory access requests from the CPU, resulting from last-level cache (LLC) misses or write-backs. The MC operates the memory chips to fetch and store data, as well as refresh their memory arrays (in the case of DRAM). Operating the memory chips entails forwarding physical addresses to the peripheral circuitry, and issuing commands to drive the chips while respecting certain timing constraints. Due to its central role, an MC can be designed with varying complexity to optimize the memory system for different purposes. For example, to improve access throughput, the MC may choose to buffer particular requests (e.g., LLC write-backs) and/or reorder requests according to the system's current state [3, 17, 30].

At a high level, a memory chip can be logically viewed as an array of bit cells with some interface circuitry. Memory chips are read and written in groups called ranks. A rank can transfer a number of bits equal to the memory channel's width (e.g., 64 bits) in one memory cycle, so it takes several memory cycles to transfer an entire LLC line. At a low level, the memory chips are subdivided into banks that can be accessed concurrently, and further into bitcell arrays. On a read to a bank, part of the target address (provided by the MC) is used to activate a row of bitcells, and amplify and latch their contents into a row buffer. Then, a column (subset of a row) can be selected, causing its data to be transferred from the row buffer to the MC. The MC then delivers the data to the CPU. An access to a column of another row causes the activation of the new row. On a LLC write-back, the data flows in the opposite direction, is stored in the row buffer, and eventually written to the memory array.

2.1 Synchronous Dynamic RAM (DRAM)

A DRAM cell is a transistor-capacitor pair (1T/1C) where each transistor allows reading and writing, and the capacitor stores a bit as an electrical charge (Figure 1). Since DRAM's capacitors discharge over time, they need to be refreshed at regular intervals to prevent data loss. In addition, reading a DRAM cell destroys its original content; the "precharge" operation restores the data from the row buffer into the array. Under a close-page management scheme, the MC precharges (closes) a row after every column access, unless there is another pending access for the same row. If there is a pending access, the row buffer remains open and the access can be serviced from there. Under open-page management, the row buffer remains open until another row needs to be loaded into it; only at that point is the precharge operation performed.

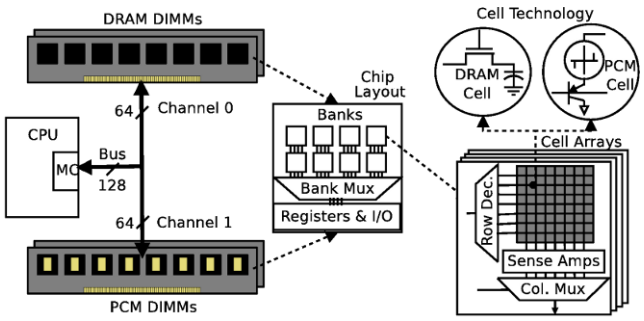

Figure 1: Memory system using DRAM and PCM DIMMs. DRAM and PCM arrays are slightly different (cells and sense amplifiers).

Close-page management typically works better than open-page management for multi-core systems [31].

Double Data Rate 3 (DDR3) is the current standard for Synchronous DRAM interfaces. DDR3 channels are 64-bit wide (72-bit wide with ECC), so it takes four memory cycles (8 transfers) to access a 64-byte LCC line. In server systems, typically one, two, or four ranks of 8 (x8) or 16 (x4) chips (plus additional chips when using ECC) are laid out on each DIMM. When designed to operate at high frequencies, only one or two DDR3 DIMMs can lie on a single memory channel due to electrical limitations.

DRAM energy can be broken down into background, activation/precharge, read/write, and termination energies. Background energy is independent of activity and is due to the peripheral circuitry (e.g., row decoder, column muxes, sense amplifiers, and bus drivers), transistor leakage, and refresh operations. The activation/precharge energy is due to these two operations on the memory arrays. The read/write energy is due to column accesses to row buffers. The termination energy is due to terminating data and strobe signals at each chip, as well as signals of other ranks on the same channel. The three latter classes are often referred to as "dynamic DRAM energy". Most of the energy consumed by DRAM is background and activation/precharge energy.

2.2 Phase-Change Memory (PCM)

A PCM cell comprises an NMOS access transistor and a storage resistor (1T/1R) made of a chalcogenide alloy. With the application of heat, the alloy can be transitioned between physical states with particular resistances, used to represent binary values. When the alloy is heated to a very high temperature ($> 600^oC$) and quickly cooled down, it turns into an amorphous glass with high electrical resistance, representing 0. When the alloy is heated to a temperature between the crystalization (300^oC) and melting (600^oC) points and cools down slowly, it crystalizes to a state with lower resistance, representing 1. This programming process can be carried out by peripheral write drivers.

A cell's content can be read using current sense amplifiers to measure its electrical resistance, as opposed to DRAM's slower but smaller voltage sense amplifiers. PCM can thus interface with most CMOS peripheral circuitry used in DRAM [37]. Unlike in DRAM, PCM's reads are non-destructive and cells can retain data for several years. On the downside, PCM cells exhibit worse access performance than DRAM.

Regarding density, PCM has similar cell size ($4F^2$ to $12F^2$) compared to DRAM ($6F^2$ to $8F^2$). However, PCM enables manufacturing of multi-level cells (MLC), which produce intermediate resistances (the alloy is partially crystalline and partially amorphous) and therefore can store multiple bits. Current MLC pro-

totypes have two- or four-bit cells, capable of storing four and sixteen binary values, respectively [24]. Assuming the same cell size for both technologies, these MLCs hold twice and eight times more data than DRAM cells in the same area.

Our assumptions for PCM. To evaluate MLCs in the middle of this range of storage densities, we assume three-bit MLCs for the PCM array. This assumption makes PCM four times more storage-dense than DRAM, as in [28]. For easier adoption, we expect that the peripheral circuitry for PCM (e.g., row buffers, row and column decoders, DIMM interface) will be equivalent to that for DRAM, except for sense amplifiers. Thus, we assume this circuitry to have the same performance and power characteristics for both PCM and DRAM. Previous papers have made the same assumption [20, 37]. Only the written cache lines in a row buffer are written back to the PCM cell array (DRAM needs the entire buffer to be written back to precharge its cells). Similar optimizations have been used before as well [20, 38]. To expose the entire overhead of PCM accesses to the cores, we study a CPU with in-order cores and a single outstanding miss per core.

PCM does not require cell refreshing or precharging, thereby lowering background energy relative to DRAM and eliminating precharge energy. However, PCM increases activation and termination energies, since its activations (actual accesses to memory cells) are slower than with DRAM. Our assumptions for peripheral circuitry imply that row buffer read/write energy is the same for DRAM and PCM.

3. HYBRID MAIN MEMORY DESIGN

Given the speed of DRAM and the high density of PCM, there is a clear incentive for combining these two technologies into a single, hybrid memory system. However, PCM has undesirable characteristics (poor performance, dynamic energy, and endurance) that must be properly managed. Similarly, DRAM has a relatively high idle energy consumption compared to that of PCM.

A few previous works have realized the benefits of combining DRAM and PCM [28, 37] and the associated tradeoffs. Unfortunately, the previous approaches have serious limitations. Qureshi *et al.* [28] proposed to use DRAM as a buffer in front of PCM. Since the buffer is managed as an inclusive hardware cache, the DRAM space does not add to the overall memory capacity. More importantly, for workloads with poor locality, the cache actually lowers performance and increases energy consumption. Zhang *et al.* [37] combine the DRAM and PCM areas in a large flat memory and migrate pages between the areas. However, the migrations are performed by the OS and target the frequently written pages, leaving read-intensive pages in the slower PCM area.

Next, we describe our hardware-software page placement policy, called RaPP (Rank-based Page Placement), which manages pages without the limitations of previous works. After that, we detail our implementation of the two prior hybrid systems. Throughout our descriptions, we differentiate between *virtual memory* pages (or simply pages) and *physical memory* frames (or simply frames).

3.1 Rank-based Page Placement

Given the characteristics of DRAM and PCM, RaPP seeks to (1) place performance-critical pages and frequently written pages in DRAM, (2) place non-critical pages and rarely written pages in PCM, and (3) spread writes to PCM across many physical frames.

The justification for these goals is that previous works have shown that typically only a relatively small subset of pages is performance-critical during the execution of a workload [15]. This observation suggests that (a) this subset may fit entirely in the DRAM part of the hybrid memory, and (b) the majority of lightly accessed pages should consume little energy, if stored in the PCM part. Moreover, previous work has found that the subset of critical pages may change over time, along with the criticality of individual pages [15]. This second observation suggests that the system must dynamically identify the critical pages and adjust their placements accordingly.

Since the OS is not on the path of most memory accesses, RaPP must be collaboratively executed by the OS and the MC. An interesting challenge is that neither the MC nor the OS has complete information about the performance criticality of the pages in a workload. For example, the latency of the cache misses associated with a page may be hidden behind out-of-order execution or multithreading by the processor cores. Interestingly, previous work [4] has shown that the frequency of cache misses is a very good proxy for a thread's performance criticality, regardless of the details of the microarchitecture (in-order vs out-of-order execution). Thus, pages that experience more misses also tend to be more performance critical.

RaPP relies on the MC to monitor the misses in the LLC to each physical memory frame. In addition, the MC monitors the LLC write-backs directed to each frame. Using this information, RaPP dynamically ranks frames based on frequency and recency of accesses, as detailed below. Frames that rank high are called "popular", and frames that rank low are called "unpopular". Whenever the most popular PCM frame reaches a threshold number of accesses (called the "migration threshold"), the MC considers migrating its content into DRAM transparently to the OS. If the DRAM area is full, the MC selects the page stored in an unpopular DRAM frame to migrate to PCM.

Ranking frames. RaPP simultaneously considers frame access frequency and recency in its dynamic ranking of pages (i.e., the pages stored in the frames), using a modified version of the Multi-Queue (MQ) [39] algorithm for second-level buffer cache replacements. As originally designed, MQ defines M LRU queues of block descriptors, numbered from 0 to $M-1$. Each descriptor includes the block number, a reference counter, and a logical expiration time. The descriptors in queue $M-1$ represent the blocks that are most frequently used. On the first access to a block, its descriptor is placed in the tail of queue 0. In addition, the block's expiration time $ExpirationTime$ is set to $CurrentTime + LifeTime$, where both times are measured in number of accesses and $LifeTime$ specifies the number of consecutive accesses that must directed to other blocks before we expire the block. Every time the block is accessed, its reference counter is incremented, its expiration time is reset to $CurrentTime + LifeTime$, and its descriptor is moved to the tail of its current queue. The descriptor of a frequently used block is promoted to a higher queue (saturating at queue $M-1$, of course) after a certain number of accesses to the block. Specifically, if the descriptor is currently in queue i, it will be upgraded to queue $i+1$ when its reference counter reaches 2^{i+1}. Conversely, MQ demotes blocks that have not been accessed recently. On each access, the descriptors at the heads of all M queues (representing the LRU block of each queue) are checked for expiration ($CurrentTime > ExpirationTime$). If a block descriptor expires, it is placed at the tail of the immediately inferior queue, and has its expiration time again set to $CurrentTime + LifeTime$.

We use the modified MQ to rank memory frames (it was originally designed to rank disk blocks). We do so for two main rea-

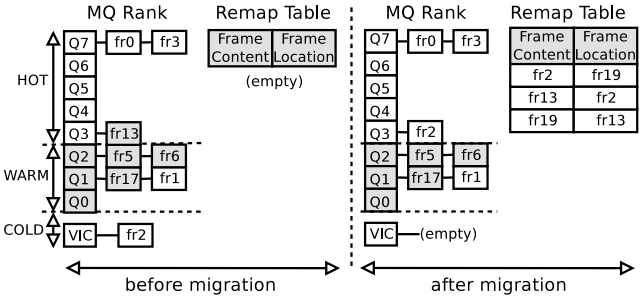

Figure 2: RaPP example. Before and after we migrate a PCM page (stored in frame 13) that crossed over to the DRAM side of the ranking to a frame from the victim list (frame 2). Dark shading represents PCM frames and no shading represents DRAM frames.

sons: (1) as page migrations are expensive operations, it is important to select the pages to migrate as intelligently and accurately as possible. MQ has been proven superior to other algorithms in selecting the blocks to replace [39]; (2) modern memory controllers are becoming increasingly complex and sophisticated (as discussed below), as a result of the increasing importance of the memory system (in terms of performance and energy) and relentless technology scaling. To avoid performance degradation, *the updates to the MQ queues are performed by the MC off the critical path of memory accesses,* using a separate queue of updates and a small on-chip SRAM cache. To find the MQ entry of a frame, the MC hashes the corresponding frame number.

We create 15 queues (numbered 0–14) plus a 16th victim list (described below). Pages stored in PCM frames that become popular (i.e., get to higher queues) are scheduled for migration to DRAM. However, we modified MQ in two important ways. First, instead of counting all accesses, we only count an access if it occurs more than a threshold time (measured in memory cycles) after the last access to the same frame. This latter threshold is called the "filter threshold". The MC stores the time of the last access in the descriptor for the frame. The reason for filtering rapid-fire accesses out is that there is no point in trying to migrate a page that is accessed in such a way; before we get to migrate the page, the needed data has already been loaded to the LLC (or evicted from it). In fact, it is possible that the page will not even be accessed again in memory. Using a 2-competitive approach, we set the filter threshold to be $MigrationCost/MigrationThreshold$, where $MigrationCost$ is the uncontended number of memory cycles needed to migrate a page. ($MigrationCost$ is roughly 1.6μs in our experiments.)

Second, we modified the demotion policy in the following ways: (a) we use time, not number of accesses, as the metric for demotion to reduce space requirements (in our experiments, we set $LifeTime$ to 100μs, which works well for our workloads); (b) we only demote from one queue at a time (in round-robin fashion) to reduce runtime overhead; and (c) a DRAM frame that is demoted twice without any intervening accesses leaves the MQ queues and becomes a candidate to receive a popular PCM page. The reason for the latter modification is that frames that undergo multiple demotions tend to have already been cached in the LCC and will not be accessed in a while. We store the MQ queues and the victim list in the lowest DRAM addresses.

Migrating pages. As mentioned above, RaPP schedules the page stored in a PCM frame for migration to DRAM after its reference counter reaches the migration threshold. In particular, the page stored at any PCM frame that reaches queue 5 (i.e., the reference counter for the frame has reached $2^5 = 32$) is scheduled for migration to DRAM. (Thus, the maximum number of frames that can be in queues 5–14 is the size of DRAM. For symmetry, the maximum size of queues 0–4 is also set to the size of DRAM.)

We find these values for M and the migration threshold to work well for our extensive set of workloads. The rationale is that in many workloads, a large number of pages would end up in queue $M-1$ if M is small, compromising the accuracy of the hot page identification. On the other hand, if M is high, RaPP can correctly identify hot pages, but the MQ overhead increases. As for the migration threshold, we must select a value that enables early migrations but without migrating pages unnecessarily.

To select a destination DRAM frame for a page, the MC maintains an LRU list of victim DRAM frames. The victim frames are not in any of the LRU queues (the list is initialized with all DRAM frames). Whenever a frame on the victim list is accessed, it is removed from the list and added to queue 0. A frame demoted from queue 0 or demoted twice without intervening accesses is moved to the tail of the list. The destination DRAM frame is the first frame on the list. If the list is empty, no destination is selected and the migration is delayed until a frame is added to the list.

To effect a page migration to DRAM, the MC (1) migrates the page stored in the selected DRAM frame to one of the unranked PCM frames, (2) migrates the content of this latter frame to the most popular PCM frame, and finally (3) migrates the content of the most popular PCM frame to the selected DRAM frame. Figure 2 shows an example, where shaded frames are PCM frames and non-shaded frames are DRAM frames. In the example, the MC migrates the content of frame 13 to frame 2, the content of frame 2 to frame 19, and the content of frame 19 to frame 13.

To allow the three migrations to proceed concurrently, the MC uses three intermediate frame-size buffers located in the MC itself. The contents of the frames are first copied to the buffers, and only later copied to the destination frames. In addition, to avoid excessively delaying LLC misses due to row conflicts while migrating, the PCM DIMMs are equipped with an extra pair of row-buffers per rank, used exclusively for migrations. Operated by the MC, these buffers communicate with the internal prefetching circuitry of the PCM DIMM [11, 12], bypassing the original bank's row buffer. Since our migrations occur in sequence, two of these buffers are necessary only when the migration involves two banks of the same rank, and one buffer would suffice otherwise. This modification is not applied to DRAM DIMMs to avoid their redesign. (The energy and delay costs of these extra PCM DIMM buffers are taken into account in our simulations.)

RaPP uses a different destination unranked (and thus unpopular) PCM frame every time it needs to migrate a page out of DRAM. The reason is that migrations involve writes to the PCM cells. Using different unpopular pages guarantees that these writes are evenly spread across the PCM area for wear leveling. We start picking unranked frames from the bottom of the physical address space (which maps to the end of the PCM area), and move upward from there whenever a new PCM frame is needed.

The set of scheduled migrations is maintained in a list. We deschedule migrations whenever the corresponding PCM pages cross back down to queue 4 before the migrations start. The MC performs migrations from the list whenever there are no LLC misses or write-backs to perform. Any misses that arrive for a page undergoing a migration are directed to the original address or to one of the intermediate buffers. Write-backs are buffered until the migration is concluded.

For best performance, our goal is to execute the migrations completely in the background and without OS involvement. Thus, the MC maintains the $RemapTable$, a hash table for translating frame addresses coming from the LLC to actual remapped frame addresses. Figure 2 shows an example $RemapTable$. The $RemapTable$ is accessible by the OS as well. Periodically or when the $RemapTable$ fills up (at which point the MC interrupts the CPU), the OS commits the new translations to its page table and invalidates the corresponding TLB entries. (When non-virtually addressed hardware caches are used, some lines may have to be invalidated as well.) We assume that the OS uses a hashed inverted page table, as in the UltraSparc and PowerPC architectures, which considerably simplifies the commit operation. Since a commit clears the $RemapTable$, the OS sets a flag in a memory-mapped register in the MC to make sure that the MC refrains from migrating pages during the commit process.

The $RemapTable$ also includes two bits for communication between the MC and the OS. One bit is called $MigratingNow$, which when set means that the corresponding frame is currently scheduled for a migration. The other bit is called $ReplacingNow$, which when set means that the OS is replacing the page currently stored in that frame. The MC is responsible for $MigratingNow$, whereas the OS is responsible for $ReplacingNow$. Before the OS tries to replace a page, it must check the $RemapTable$ first. There are three possible scenarios here. Scenario 1: If there is no entry for the physical frame in which the page lies, the OS creates one, sets $ReplacingNow$, and programs the DMA engine to use the frame. The MC does not migrate any page to that same frame while $ReplacingNow$ is set. When the replacement is done, the OS resets $ReplacingNow$. Scenario 2: If there is an entry for the corresponding frame and $MigratingNow$ is set, the OS should select another page for replacement. Scenario 3: If the frame has already changed addresses (i.e., the entry for the frame exists and $MigratingNow$ is not set), the OS can set $ReplacingNow$ and proceed using the new frame address.

Finally, for robustness, RaPP uses a self-disabling mechanism that disables access monitoring, queue maintenance, and migrations whenever too many "bad migrations" occur. A bad migration occurs in one of two cases: (1) when a page originally in PCM is migrated to DRAM and then back to PCM without being referenced enough times while in DRAM; or (2) when a page originally in DRAM is evicted to PCM and then back to DRAM with too many accesses while in PCM. To implement this mechanism, we use a single counter of bad migrations (CBM) and a 2-bit saturating counter per MQ entry. Whenever a ranked page is touched, the saturating counter is incremented. Whenever a migration is completed, using the $RemapTable$, RaPP can identify where the page was since the last commit to the OS page table. If the migration falls into case (1) and the counter is not saturated, or it falls into case (2) and the counter is saturated, CBM is incremented. The saturating counter for a page is reset whenever the page migrates. At the end of each 1ms epoch, if the number of bad migrations reached 5% (the "disable threshold") or more of the maximum number of migrations possible within the epoch, RaPP is disabled.

Controller structure. Our MC adds a few components to a vanilla MC. Our MC (Figure 3) extends a programmable MC from [34] by adding RaPP's own modules (shaded in the figure). The MC receives read/write requests from the LLC controller via the CMD queue. The Arbiter iteratively dequeues requests, which the Controller converts into commands to be sent to the memory devices.

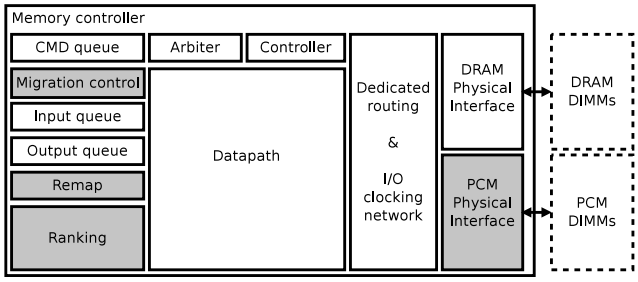

Figure 3: Memory controller with RaPP's new modules highlighted.

The Controller routes those commands by frame address to the appropriate Physical Interface (DRAM or PCM), which converts commands into timing relationships and signals for operating memory devices and coordinating data transfers. The Interfaces also control the power states of their respective memory ranks. The Datapath handles the data flow from the memory devices to the LLC controller and vice-versa. The module places data read from memory into the Output queue, where the LLC controller can read it. On write-back requests, the Datapath reads data (provided by the LLC controller) from the Input queue. For consistency, the CMD queue logic checks if the target address of a read or write-back collides with older write-backs in the Input queue. A colliding read is serviced from the queue without actually reaching the memory devices, thus finishing faster. A colliding write-back invalidates the older write-back command and data.

RaPP's Ranking module (RKMOD) contains the small on-chip cache of MQ and victim entries (entry cache) and the queue for updates to the MQ queues and the victim list (update queue). Misses in the entry cache produce requests to DRAM. RKMOD's logic snoops the CMD queue, creating one update per new request. To reduce the lag between an access and its corresponding MQ entry update, the update queue is implemented as a small circular buffer (32 entries), where an entering update precludes any currently queued update to the same entry.

The Migration Control module (MIGMOD) contains the queue of scheduled migrations (migration queue) and three page-sized buffers for the migrations (transfer buffers). MIGMOD processes migrations sequentially, each one occurring in two stages: (1) read and buffer frames; and (2) write frames to their new locations. Stage (2) does not start until stage (1) is completed. MIGMOD latches the base addresses of the frames undergoing a migration, as well as an offset address within each frame. The Controller module detects memory accesses that target one of the frames undergoing migrations by checking the base addresses. The Controller serves an access that targets a migrating frame by accessing the appropriate structure (main memory or a transfer buffer).

The Remap module (REMOD) contains the $RemapTable$ and the logic to remap target addresses. At the end of a migration, MIGMOD submits the three latched frame numbers to REMOD, which creates new mappings in the $RemapTable$. REMOD snoops the CMD queue to check if it is necessary to remap its entries. Each $RemapTable$ lookup and each remapping take 1 memory cycle. However, these operations only delay a request if it finds the CMQ queue empty.

Note that many previous works have proposed MCs with similar levels of sophistication and complexity, e.g. [10, 17, 18, 25, 36, 37, 13]. For example, [17] implements a learning algorithm in the memory controller itself.

Storage overhead. The bulk of our MC design is in the storage

structures that it contains. The total on-chip storage in our design is 126 KBytes. By design, the page buffers require 24 KBytes (3 pages). The other structures have empirically selected sizes: 28 KBytes for the $RemapTable$ (4K entries), 64 KBytes for the cache of MQ and victim entries (4K entries), and 10 KBytes for the update and migration queues. This amount of on-chip storage is small compared to the multi-MByte shared LLC.

Our design also has limited DRAM space requirements. Taking a system with 1GB of DRAM + 32GB of PCM as a base for calculation, the total DRAM space consumed by descriptors is 6 MBytes (0.59% of the DRAM space). Each frame descriptor in the MQ queues or in the victim list takes 124 bits, which we round to 128 bits. Each descriptor contains the corresponding frame number (22 bits), the reference counter (14 bits), the queue number, including victim list (4 bits), the last-access time (27 bits), three pointers to other descriptors (54 bits), a flag indicating that the frame has been demoted (1 bit) and the counter for bad migrations (2 bits). For the configurations with which we experiment, the space taken by the descriptors is 0.63 MBytes.

3.2 Comparable Hybrid Memory Systems

We compare our design to the two most closely related hybrid memory systems: DBUFF [28] and WP [37].

DRAM Buffer (DBUFF) relies on a DRAM buffer logically placed between the CPU and a main memory composed solely of PCM [28]. The DRAM buffer is implemented as a set-associative cache managed entirely by the MC and invisible to the OS. Cache blocks (corresponding to virtual memory pages) coming from secondary storage are initially installed in the DRAM buffer, but also take space in PCM. From then on, the memory accesses are directed to the DRAM buffer. On a buffer miss, the page containing the desired cache line is brought into the buffer from PCM. When a block is replaced from the buffer (using the clock algorithm), it is written to its PCM frame if this is the first eviction of the block or the block was written in the buffer. Block writes to PCM are enqueued in an write buffer and done lazily in background. Like in our design, only the cache lines that were actually written are written to PCM.

When workloads exhibit good locality, most accesses hit the DRAM buffer, which leads to good performance and dynamic energy. Endurance is also good since the lazy block writes and cache-line-level writes substantially reduce the write traffic to the PCM array. (In fact, our implementation of DBUFF does not include the Fine-Grained Wear Leveling and Page-Level Bypass techniques proposed in [28]. The reason is that the endurance produced by the other techniques is sufficient in our experiments; adding extra complexity does not seem justified for our workloads and endurance assumptions.) However, workloads with poor locality may lead to poor performance and energy consumption. In addition, the inclusive DRAM caching in DBUFF reduces the amount of available memory space, potentially leading to a larger number of page faults than our design.

Our simulations of DBUFF are optimistic in many ways. First, we consider a DRAM buffer of size approximately 8% of the total memory size (rather than the original 3%). Second, we assume no DRAM buffer lookup overhead in performance or energy. Third, we implement the DRAM buffer as a fully associative structure (rather than set associative) with LRU replacement (rather than clock). Fourth, on a DRAM buffer miss requiring a page write-back, the dirty blocks (only) are written back at the same time as the missing page's content is fetched from PCM or disk.

Despite these optimistic assumptions, RaPP improves on DBUFF in two fundamental ways: (1) it uses the entire memory as a flat space, relying on page migration rather than replication; and (2) it detects when most migrations are useless and turns itself off.

Hot-modified Pages in Write Partition (WP) places DRAM and PCM in a flat address space and treats DRAM as an OS-managed write partition [37]. All pages are initially stored in PCM. The idea is to keep the cold-modified (infrequently written) pages in PCM, trying to take advantage of its low idle power consumption, and the hot-modified (frequently written) pages in DRAM to avoid PCM's high write latency and poor endurance. The MC implements a variation of the MQ algorithm with 16 LRU queues, but only counts write accesses to the physical frames. Frames that reach queue 8 (receive 2^8 writes) are considered to store hot-modified pages. On a page fault, the OS brings the page from secondary storage to the PCM area. Over time, the pages that become hot-modified are migrated to DRAM by the OS. At the same time, a page currently in DRAM but with fewer writes may have to be migrated back to PCM.

Our simulations of WP are also optimistic, as we do not charge any performance or energy overheads for the data structures and hardware modifications necessary to implement WP.

Despite these optimistic assumptions, there are three main problems with WP. First, it can hurt the performance of read-dominated workloads under less optimistic assumptions about PCM read performance. Second, migrating pages using a core at the OS quantum boundary wastes opportunities to improve performance and energy-delay within that timespan. Third, endurance also suffers because it takes a large number of writes until the OS will consider migrating a heavily written page to DRAM. Our evaluation studies mainly how WP compares to other approaches. However, in our longer technical report [29], we also isolate the impact of migrating frequently-read pages and enabling migrations within the OS quantum via hardware.

RaPP improves on WP by: (1) migrating pages that are read-intensive as well; (2) migrating pages in the background, without OS involvement; (3) including mechanisms for identifying pages worthy of migration and self-disabling for when migrations are mostly useless; and (4) spreading migrations across many physical frames. Moreover, this paper improves on [37] by: (5) assuming more realistic PCM characteristics; and (6) presenting a comparison of RaPP and WP to DBUFF, across a large set of workloads and parameters. We study variations of WP in [29].

4. EVALUATION

In this section, we evaluate hybrid memory systems using energy and performance as first-order metrics. Although we also report endurance results, we give them lower emphasis because our system can be easily combined with many previously proposed techniques to mitigate the PCM endurance problem (Section 5).

4.1 Methodology

Our evaluation is based on simulation, since PCM hardware is not yet available. We simulate combinations of benchmarks from the SPEC 2000, SPEC 2006, and Stream suites forming a total of 27 workloads (Table 1). Because our workloads have widely different memory footprints, we group them with respect to footprint size into Large (LG), Medium (MD), and Small (SM) classes.

To reduce simulation times, our simulations are done in two steps. In the first step, we use M5 [5] to collect memory access (LLC misses and write-backs) traces from our workloads running on an 8-core server. Each benchmark in a workload is represented

Table 1: Workload described by tag, memory footprint in MB (Foot), LLC misses per 1000 instructions (MKPI), and percentage of LLC write-backs as a fraction of all memory accesses (WB%). *Spec 2006.

Tag	Foot	MKPI	WB%	Applications (x2 each)
LG1	993	12	33	milc*, gobmk*, sjeng*, libquantum*
LG2	992	29	32	S.add, S.copy, apsi, milc*
LG3	746	24	27	mcf*, S.triad, sjeng*, facerec
LG4	743	4	25	vortex, milc*, sixtrack, mesa
LG5	702	24	26	sjeng*, S.triad, S.add, swim
LG6	683	4	28	perlbmk, crafty, gzip, milc*
LG7	645	25	32	lucas, gcc, mcf*, sphinx3*
LG8	594	18	32	wupwise, vpr, mcf*, parser
LG9	557	17	32	swim, eon, art, lucas
MD1	486	13	49	applu, lucas, gap, apsi
MD2	467	23	32	S.scale, S.triad, swim, eon
MD3	414	20	30	mcf*, parser, twolf, facerec
MD4	407	8	24	namd*, S.triad, sjeng*, wupwise
MD5	394	13	32	art, lucas, mgrid, fma3d
MD6	385	24	28	art, mcf*, gzip, vpr
MD7	381	14	23	S.add, h264ref*, equake, hmmer*
MD8	367	46	33	S.triad, S.add, S.copy, S.scale
MD9	356	30	27	equake, S.scale, S.triad, mgrid
SM1	295	2	21	wupwise, gobmk*, vortex, h264ref*
SM2	285	5	33	swim, perlbmk, namd*, eon
SM3	283	6	33	swim, crafty, twolf, gcc
SM4	276	16	33	lucas, h264ref*, libquantum*, sphinx3*
SM5	271	15	27	wupwise, equake, ammp, libquantum*
SM6	260	11	24	fma3d, mgrid, galgel, equake
SM7	247	12	32	fma3d, sphinx3*, galgel, lucas
SM8	243	15	21	S.triad, h264ref*, fma3d, equake
SM9	243	2	29	ammp, gap, wupwise, vpr

by its best 100M-instruction simulation point (selected using Simpoints 3.0 [26]). A workload terminates when the slowest application has executed 100M instructions.

In the second step, we replay the traces using our own detailed memory system simulator. This simulator models all the relevant aspects of the OS, memory controller, and memory devices, including inverted page tables and TLB management, page replacements, memory channel and bank contention, memory device power and timing, and row buffer management.

The main architectural characteristics of the simulated server are listed in Table 2. We simulate in-order cores to expose the overheads associated with PCM accesses to workloads. The cores have private 64-Kbyte 2-way instruction and data L1 caches, as well as an 8-MByte 8-way combined shared cache. For this cache architecture, Table 1 reports the LLC misses per kilo instruction (MPKI) and the percentage of LLC write-backs (WB%) for each workload. The memory system has 4 DDR3 channels, each one occupied by a single-rank DIMM with 8 devices (x8 width) and 8 banks per device. In all simulations, we assume an initially warm memory (no cold page faults). The MC implements cache-block-level bank interleaving and page-level channel interleaving. Memory accesses are served on a FCFS basis. The MC uses close-page row buffer management. (More sophisticated access scheduling is not necessary for our simulated system and workloads, as opportunities to increase their bank hit rate via scheduling are rare, and such improvements are orthogonal to our study.)

A memory rank can be in (1) Active Standby state, when at least one of its banks is serving requests; or (2) Precharge Power Down, when all banks are idle and the clock enable line is turned off to save energy. Additionally, PCM is enhanced to avoid writing unmodified cache lines back to the cell array. The table shows power parameters [20] of DRAM and PCM chips, and the timing parameters that change across memory technologies [20, 28, 37] (although we simulate all relevant parameters [22, 32]).

Besides RaPP, we simulate the two hybrid approaches mentioned in Section 3 (DBUFF and WP) and an additional "Unmanaged" system, in which pages remain in the frames originally assigned to them by the OS. We use Unmanaged as the baseline for comparison. *Only RaPP is assumed to have energy and performance overheads stemming from its data structures.* Our assumptions for RaPP are consistent with those of other authors [14]. For example, the RaPP SRAM consumes 0.13W of background power and 0.017nJ per 16-byte operation; the transfer buffers consume 0.06W of background power and 0.06nJ per 64-byte operation; and each DIMM-level row buffer increases the rank background power by 12.5% when active (and as much dynamic power as a regular row buffer). The four hybrid systems have 1 channel equipped with 1 DRAM DIMM (128MB) and the remaining 3 channels with 1 PCM DIMM each (3x128x4MB=1536MB), totaling 1.664 GBytes of memory. We picked these small memory sizes to match the footprint of the workloads' simulation points.

Table 2: System settings.

Feature		Value	
CPU cores (2.668GHz, Alpha ISA)		8 in-order, one thread/core	
TLB per-core size, hit/miss time		128 entries, 8/120 CPU cycles	
L1 I/D cache (per core)		64KB, 2-way, 1 CPU cycle hit	
L2 cache (shared)		8MB, 8-way, 10 CPU cycle hit	
Cache block size / OS page size		64 bytes / 8KB	
Memory (667MHz/DDR3-1333)		8KB rows, close-page	
Memory devices (x8 width, 1.5V)		DRAM	PCM
Delay	tRCD	15ns	56ns
	tRP	15ns	150ns
	tRRDact	6ns	5ns
	tRRDpre	6ns	27ns
	Refresh time	64ms	n/a
	tRFC / tREFI	110ns / 7.8μs	n/a
Current	Row Buffer Read	200mA	200mA
	Row Buffer Write	220mA	220mA
	Avg Array R/W	110mA	242mA
	Active Standby	62mA	62mA
	Precharge Powerdown	40mA	40mA
	Refresh	240mA	n/a
Normalized Density		1	4
Data Retention		64 ms	$>$ 10 years
Cell endurance (writes)		$> 10^{16}$	$10^8 - 10^9$

As another basis for comparison, we use a PCM-only system with 2 GBytes of memory (4x128x4MB). Previous works have shown that the DRAM-only system exhibits much worse performance than PCM-only and hybrid systems [28], due to its lower storage capacity, so we do not consider it in this paper.

4.2 Results

4.2.1 Performance and energy

We now compare the behavior of RaPP, DBUFF, WP, and the two baseline systems. Due to space limitations, we do not present separate performance and energy results for all workloads; instead, we plot these results for the MD workloads and discuss the results for other workloads in the absence of figures. Later, we plot energy-delay^2 (ED2) results for all workloads.

Figure 4 presents the running time (top graph, including the geometric mean of the MD results), average memory power consumption (middle graph, including the power consumed by the SRAM cache used in RaPP), and number of page migrations in RaPP and WP and page fetches from PCM in DBUFF (bottom graph, again including the geometric mean). We refer to page migrations and fetches collectively as page transfers. The performance and power results are normalized to Unmanaged. Note that we plot average power, rather than energy, to remove the impact of different running times (which are plotted in the top graph).

Running time. The top graph shows that RaPP exhibits the low-

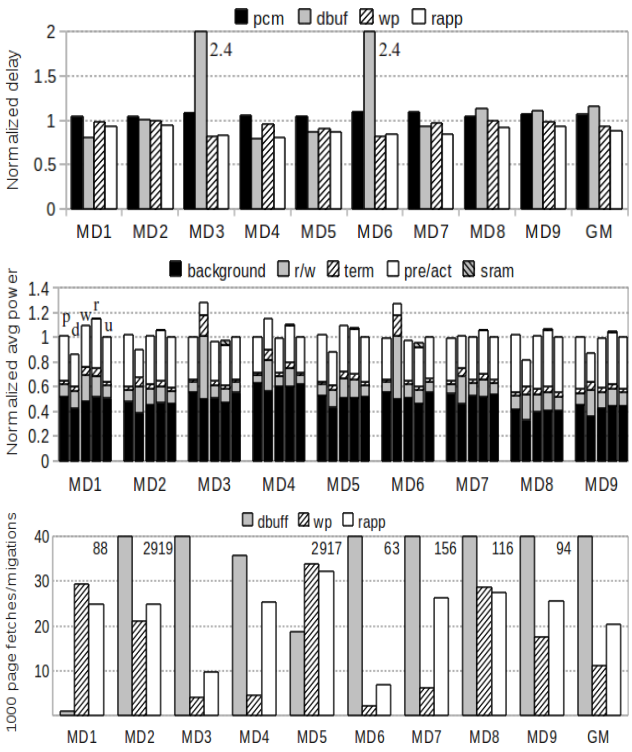

Figure 4: Comparing performance, average power, and page transfers for the MD workloads. In the middle figure, p = PCM-only, d = DBUFF, w = WP, r = RaPP and u = Unmanaged.

est average running times, followed by WP, Unmanaged, PCM-only, and then DBUFF. In fact, RaPP performs better than PCM-only and Unmanaged for all workloads (including the LG and SM workloads). RaPP achieves these results by migrating popular pages to the DRAM area, which has substantially better performance in the presence of row buffer misses than PCM. WP and DBUFF do not always outperform PCM-only and Unmanaged. RaPP is more robust than WP and DBUFF, achieving good performance in most cases and preventing degradation in others by disabling itself.

WP attempts to migrate pages to DRAM as well, but focuses solely on those pages that experience a large number of write-backs (or writes of disk data read into memory). In addition, the fact that WP migrates pages at the end of the OS quantum has two sides. On the negative side, WP misses opportunities to migrate popular pages as their popularity shifts within the OS quantum. MD2 and MD8, for example, suffer from this problem. On the positive side, migrating infrequently reduces the number of migrations in workloads where most migrations will turn out useless. However, RaPP is more effective than WP at preventing unnecessary migrations, as it includes mechanisms designed explicitly for this purpose. For example, in SM2, a large fraction of pages is hot-modified, but performance-irrelevant. In that case, RaPP disables itself at about a quarter of the execution. A similar phenomenon occurs in 6 other workloads in our experiments. In essence, RaPP is more effective at migrating the actual performance-critical pages to DRAM, improving performance by 6% for MD workloads and 7% overall with respect to WP.

The most extreme results happen for DBUFF. It achieves the lowest running time for MD1, but dismal running times for MD3 and MD6. The reason for DBUFF's poor performance is that MD3 and MD6 exhibit poor locality (their working sets are substantially larger than the size of the DRAM buffer). The same effect occurs for several LG workloads. Poor locality forces frequent page fetching from PCM into DRAM, with the eviction of dirty blocks (if any) within victim pages done in the background. Without the problematic workloads, RaPP still performs 14% and 8% better than DBUFF for the LG and MD workloads, respectively. On the other hand, the SM workloads have working sets that easily fit in the DRAM buffer, so DBUFF performs best for 6 of them. However, RaPP's overall average performance is still slightly better than DBUFF's for the SM workloads.

Finally, note that Unmanaged consistently outperforms PCM-only. The reason is that Unmanaged benefits from accessing data in its DRAM section, which is substantially faster than accessing data in PCM when row buffer misses occur. (Excluding the effect of page transfers, the row buffer miss ratio we observe is always higher than 80%. This effect has been observed in previous studies of multi-core servers as well, e.g. [31].)

Average memory power. Considering the middle graph of Figure 4, we see that all systems exhibit similar average power consumption for the MD workloads (the same happens for the LG and SM workloads). The average power correlates well with the total number of accesses in each approach. As expected, page transfers increase read/write and activation/precharge (activation/write to array for PCM) power somewhat. We also see that DBUFF produces lower background power. The reason is that the PCM area can be in precharge powerdown state more often in DBUFF.

Interestingly, although RaPP uses an SRAM cache for its ranking of pages, the power consumed by this cache is negligible. Most rank accesses become cache hits (at least 90% in our experiments). The SRAM results in the figure account for the static and hit energies. The energy consumed by the misses is reported in the other categories.

Page transfers. The bottom graph shows that RaPP migrates more pages than WP for most (all but 3) MD workloads. The same happens for most SM (all but 1) and LG (all but 2) workloads. The reason is that each migration operation in RaPP actually transfers 3 pages, instead of 1 or 2 pages in WP. Interestingly, DBUFF fetches many more pages from PCM than RaPP and WP migrate in the MD workloads. Again, the reason is that these (and the LG) workloads have working sets that do not fit in the DRAM buffer. For the SM workloads, DBUFF fetches pages much less frequently than for the MD and LG workloads. In fact, DBUFF transfers fewer pages than WP in 4 SM workloads and fewer than RaPP in 6 of them.

Putting performance and energy together: ED2. Figure 5 plots the ED2 of each workload and system normalized to Unmanaged. The rightmost set of bars in each graph shows the geometric mean of the results presented in the same graph. The graphs show that RaPP is the only system to achieve ED2 no higher than Unmanaged and PCM-only for all workloads. (In the few instances where RaPP achieves roughly the same ED2 as Unmanaged, it disabled itself due to a large percentage of bad migrations.) Considering the workload classes independently, we find that RaPP achieves the lowest average ED2 for the LG and MD workloads. For the SM workloads, the combination of low run time, very few migrations, and idle PCM ranks (as discussed before) leads DBUFF to the lowest average ED2 (14% lower than RaPP). Over-

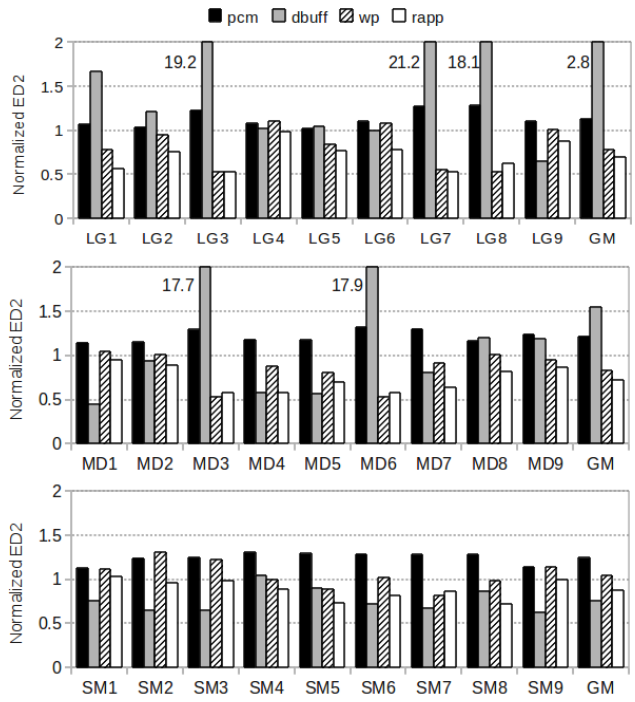

Figure 5: Comparing energy-delay^2 for all workloads.

all, RaPP achieves 13%, 24%, 36%, and 49% lower ED2 than WP, Unmanaged, PCM-only, and DBUFF, respectively.

As mentioned above, RaPP improves ED2 over PCM-only and Unmanaged by migrating popular pages from PCM to DRAM. The comparison to WP is more interesting. RaPP and WP achieve comparable ED2 for some workloads. However, WP achieves worse ED2 than Unmanaged for 2 LG, 1 MD, and 4 SM workloads. In many of these cases, RaPP did very well but in others it simply disabled itself. Compared to DBUFF, RaPP wins for workloads with working sets larger than the DRAM buffer (i.e., 9 of our workloads).

4.2.2 *Endurance*

To evaluate the system lifetimes (limited by PCM's endurance), we resort to the Required Endurance metric [6]: $T_{life} \times \frac{B}{\alpha\, C}$, where B is the memory bandwidth in bytes per second, T_{life} is the desired system lifetime in seconds, α is the wear-leveling efficiency of the system, and C is the memory capacity in bytes. For each workload, we consider T_{life} the typical 3- to 5-year server lifespan. α is A/M, where A is the average number of writes per PCM cache block and M is the number of writes to the most written cache block in the PCM area of the system. A low value of α suggests poor wear-leveling. Required Endurance determines the number of writes that a PCM cache block must be able to withstand during the server's lifespan.

Figure 6 compares the base-10 logarithm of the Required Endurance of the systems we consider, using the 3- and the 5-year projections. For comparison, PCM's endurance today is $10^8 - 10^9$. The figure shows that RaPP requires roughly 10% more endurance than DBUFF, but only 1% more than WP on average. In contrast, it requires 5% and 4% less endurance than PCM-only and Unmanaged, respectively. DBUFF provides the best Required Endurance.

In RaPP's 5-year projection, most MD and SM workloads require endurance only slightly higher than 10^9, whereas a few LG

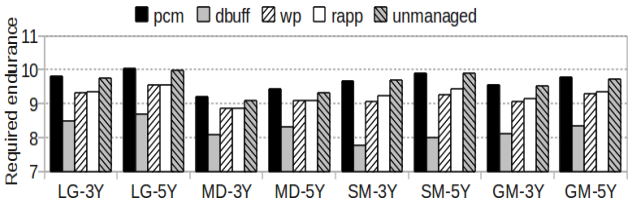

Figure 6: Required Endurance projected over 3 and 5 years.

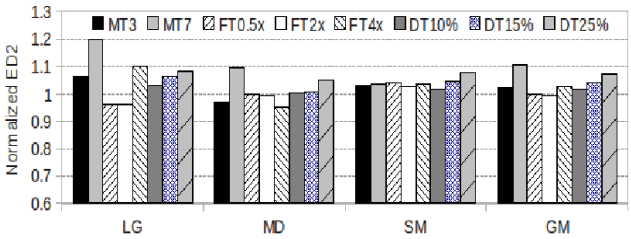

Figure 7: RaPP's sensitivity to migration threshold (MT), filter threshold (FT), disable threshold (DT).

workloads require closer to 10^{10} endurance. The reason for these results is that in RaPP the frequently read and frequently written pages compete for space in the DRAM area, trying to strike a balance between performance and endurance. In addition, in some cases, RaPP's self-disabling mechanism is triggered, making it behave as Unmanaged from that point on.

However, we argue that RaPP's endurance results are not a cause of concern for two reasons. First, as we mention in Section 5, many proposals already extend the lifetime of PCM greatly. Several of them are orthogonal to RaPP [8, 18, 20, 38] and can enhance PCM independently from our approach. Second, the predictions for future PCM cells suggest significant endurance improvement compared to DRAM (10^{12} writes by 2012 [9] and 10^{15} writes by 2022 [1]), but performance and energy will still lag. Better PCM endurance justifies a shift towards reducing energy and delay, as we do in this paper.

4.3 Sensitivity analysis

RaPP parameters. We now study the impact of the migration threshold (MT), which defines the number of accesses to a PCM page before deciding to migrate it to DRAM; the filter threshold (FT), which defines the window of time for filtering consecutive page references; and the disable threshold (DT), which defines the percentage of bad migrations before RaPP disables itself.

Figure 7 depicts the ED2 results for the three workload classes and the overall geometric mean. Each bar is normalized to the ED2 resulting from the default value of the corresponding parameter. Specifically, the default value for MT is 2^5 and we compare it to 2^3 (labeled MT3) and 2^7 (MT7); the default value for FT is $MigrationTime/32$ and we compare it to 0.5x default FT (FT0.5x), 2x default FT (FT2x), and 4x default FT (FT4x); and the default value for DT is 5% and we compare it to 10% (DT10%), 15% (DT15%), and 25% (DT25%). We always vary one parameter at a time, keeping others at their default values.

The figure shows that RaPP is most sensitive to MT, especially for the LG workloads. In particular, MT7 degrades ED2 by 20% on average for the LG workloads, compared to the default. It also degrades ED2 for the MD and SM workloads. In contrast, MT3 degrades ED2 for the LG and SM workloads, but not by as much and not for the MD workloads. RaPP exhibits relatively low sensitivity to FT, except for FT4x for LG workloads. Finally, RaPP consistently showed more ED2 degradation as DT increases.

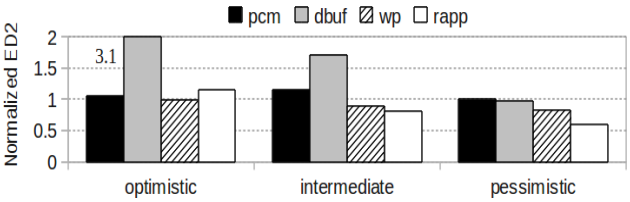

Figure 8: Sensitivity to PCM's characteristics.

These results suggest that (1) overshooting the ideal MT is more harmful on average than undershooting it; and (2) lower values for FT and DT provide better behavior. In contrast, varying these settings has a negligible impact on Required Endurance; it varies only within 1% compared to our default RaPP results.

PCM's characteristics. We now evaluate the impact of the performance and energy characteristics of PCM devices. We consider three additional settings for PCM performance: optimistic, intermediate, and pessimistic; the PCM energy varies along with its performance settings. Specifically, the optimistic setting assumes that row activations are 40% faster than our default value, bringing them close to DRAM's activations; the array writes stay with their default values. [37] assumed similarly high-performing PCM activations. The pessimistic setting assumes that activations and array writes are 25% and 2.5x slower than our defaults, respectively. [12] assumed similarly pessimistic parameters. Finally, the intermediate setting assumes that activations are 25% faster and array writes are 50% slower than our defaults.

Figure 8 shows the average ED2 results (across all workloads) normalized to Unmanaged. We can see that RaPP achieves the best average ED2 for the pessimistic and intermediate cases. The advantage of RaPP is particularly significant in the pessimistic scenario. For the optimistic setting, PCM-only and WP achieve roughly the same ED2 as Unmanaged. This is not surprising since the optimistic performance and energy of PCM become comparable to those of DRAM. Because RaPP attempts to migrate pages to DRAM despite the optimistic assumptions for PCM, it achieves slightly higher ED2. In contrast, DBUFF achieves worse ED2 because of the workloads with poor locality.

These results suggest that RaPP behaves better than the other systems for different sets of expected PCM characteristics (default and intermediate). RaPP's benefits will be even more pronounced, if commercial PCM devices turn out to be worse than our expectations. Again, the Required Endurance varies negligibly (less than 2%) across these parameter settings.

5. RELATED WORK

Using technologies other than DRAM for main memory. Despite the problems with Flash technology (page-level interface of NAND Flash, very high write and block-erase times, low cell endurance), two studies have considered combining Flash and DRAM in main memory. ENVy [33] focused on sidestepping the write-related problems of Flash using battery-backed SRAM and virtual memory techniques. A more recent position paper [23] considered a flat DRAM+Flash (or PCM) address space with the OS responsible for predicting page access patterns and migrating read-only pages from DRAM to (write-protected) Flash. Although the paper did not include an evaluation of their system, we expect that an OS-only approach to page management would cause unacceptable overhead for most types of pages.

With better characteristics than Flash, PCM is a more promising technology for use in main memory. In fact, several recent works [7, 8, 20, 27, 28, 35, 37, 38] have proposed systems that use PCM as a partial or complete replacement for DRAM. We compared our design to two of these works.

Tackling PCM's endurance problem. Many previous works focused extensively on the work-arounds needed to mitigate this problem. Lee *et al.* [20] proposed tracking data modifications at the cache block and data word levels to avoid unnecessary traffic to the MC and writes to the PCM array. The technique was combined with narrow row buffer sets organized to exploit locality.

Yang *et al.* [35] proposed the Data-Comparison Write (DCW) approach, which only allows a write to a PCM cell if the new value differs from the previously stored one. Flip-and-Write [8] improves DCW by storing extra "flip bits" that denote the encoding of each PCM word and performing hamming distance calculations to verify if reversing the encoding (by inverting flip bits) will reduce the number of writes. Alternatively, Zhou *et al.* [38] improved DCW using periodic byte rotation at the cache block level and segment swaps across memory areas.

A complementary technique named Fine-Grained Wear-Leveling (FGWL) seeks to balance the wear-out across cache blocks within a physical PCM frame by rotating blocks within the frame [28]. FGWL inspired Start-Gap [27], a technique that applies a simple algebraic function to transform a logical memory address into a physical one. [27] also combined Start-Gap with simple address-space randomization techniques.

Another approach [18] improves endurance by reusing a pair of physical frames with complementary faulty cells as a single logical frame. The system creates the pairs periodically, as new faults may alter previous pairings.

Differently than these previous systems, our work takes a higher level, page-based approach to wear leveling. First, we migrate write-intensive pages to DRAM. Second, we migrate pages coming from DRAM to unpopular PCM frames. Importantly, their low-level techniques are orthogonal and can be nicely combined with our page-based techniques to extend endurance further.

Page migration in main memory. A few works have considered page migration for memory energy conservation. Lebeck *et al.* [19] conducted a preliminary investigation of popularity-based page allocations to enable sending (unpopular) memory devices to low-power state. In [16], the OS migrates pages periodically based on their reference bits, without any support from the MC. In contrast, Pandey *et al.* [25] proposed to implement popularity-based page ranking and migration using the MC to reduce the energy consumed by DMA transfers. Dong *et al.* [13] migrate hot pages from the off-chip memory to a chip-integrated memory. In a position paper [3], Bellosa proposed "memory management controllers" (MMCs) that would take away the responsibility for memory management (e.g., page migration) from the OS.

Although an OS-only approach to page migration [16] may work for energy conservation, it does not react quickly and efficiently enough to mitigate the problems with PCM. Thus, it is critical to involve the MC (or an MMC) as well. In this context, the closest prior work to this paper is [25]. However, as our environment (a DRAM+PCM hybrid in a multiprocessing system) and goal (improve performance at the same energy consumption) are quite different than theirs (DRAM-only memory and DMA-related energy conservation in a uniprocessor system), there are many differences between the two approaches. First, we had to adapt a sophisticated ranking algorithm to address the poor performance and endurance of PCM. Pandey *et al.* used a much sim-

pler histogram-based ranking. Second, our migration approach is also more complex, as it needs to consider the limitations of PCM and involve an additional, properly selected PCM frame. Third, we considered many multiprocessing workloads and the increased pressure they put on the memory system. Because of this pressure, migrations need to be done very selectively, so we had to devise heuristics to avoid certain migrations.

6. CONCLUSIONS

In this work, we introduced a novel hybrid memory system design combining DRAM and phase-change memory. Our design features a sophisticated memory controller, and a page ranking and migration policy called RaPP. The memory controller monitors the memory accesses and implements RaPP. RaPP seeks to benefit from the best characteristics of DRAM and PCM, while avoiding their limitations. The policy takes into account the recency and frequency of page accesses, as well as the write traffic to each page, to rank pages and lay them out in physical memory. Our results demonstrate that our design behaves significantly better than two state-of-the-art hybrid designs, despite our optimistic assumptions for the latter systems.

We conclude that, although useful, existing designs are not robust to certain workloads. This robustness can be achieved by relying on page migration (rather than caching-style replication) in the background, optimizing read traffic as well as write traffic, and self-monitoring and disabling. Our controller and policy are good examples of how to achieve these characteristics.

Acknowledgments

We thank Rekha Bachwani, Kien Le, and Wei Zheng for their help in the early stages of this work. We are also grateful to Qingyuan Deng, Stijn Eyerman, Daniel Jimenez, Yoshio Turner, and Hongzhong Zheng for their help with our simulation infrastructure. We thank Abhishek Bhattacharjee for comments that helped us improve this paper. Finally, we thank our sponsors: NSF grant #CCF-0916539, Intel, and CAPES (Brazil).

7. REFERENCES

[1] Emerging Research Devices. International Technology Roadmap for Semiconductors (ITRS), 2007.

[2] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[3] F. Bellosa. When Physical Is Not Real Enough. In *ACM SIGOPS European Workshop*, 2004.

[4] A. Bhattacharjee and M. Martonosi. Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. In *ISCA*, 2009.

[5] N. Binkert et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4), 2006.

[6] G. W. Burr et al. Phase change memory technology. 2010. Journal of Vacuum Science & Technology B.

[7] A. Caulfield et al. Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing. In *SC*, 2010.

[8] S. Cho and H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *MICRO*, 2009.

[9] J. Condit et al. Better I/O Through Byte-Addressable, Persistent Memory. In *SOSP*, 2009.

[10] B. Diniz et al. Limiting the Power Consumption of Main Memory. In *ISCA*, 2007.

[11] X. Dong et al. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In *SC*, 2009.

[12] X. Dong, N. Jouppi, and Y. Xie. PCRAMsim: System-Level Performance, Energy, and Area Modeling for Phase-Change RAM. In *ICCAD*, 2009.

[13] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *SC*, 2010.

[14] X. Guo, E. Ipek, and T. Soyata. Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing. In *ISCA*, 2010.

[15] H. Huang et al. Improving Energy Efficiency by Making DRAM Less Randomly Accessed. In *ISLPED*, 2005.

[16] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *USENIX*, 2003.

[17] E. Ipek et al. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *ISCA*, 2008.

[18] E. Ipek et al. Dynamically Replicated Memory: Building Reliable Systems From Nanoscale Resistive Memories. In *ASPLOS*, 2010.

[19] A. Lebeck et al. Power Aware Page Allocation. In *ASPLOS*, 2000.

[20] B. Lee et al. Architecting Phase Change Memory as a Scalable DRAM Architecture. In *ISCA*, 2009.

[21] C. Lefurgy et al. Energy Management for Commercial Servers. *IEEE Computer*, 36(12), December 2003.

[22] Micron. 1Gb: x4, x8, x16 DDR3 SDRAM Features. http://download.micron.com/pdf/datasheets/dram/ddr3/1Gb_DDR3_SDRAM.pdf, 2006.

[23] J. C. Mogul et al. Operating System Support for NVM+DRAM Hybrid Main Memory. In *HotOS*, 2009.

[24] T. Nirschl et al. Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory. In *IEDM*, 2007.

[25] V. Pandey et al. DMA-Aware Memory Energy Management. In *HPCA*, 2006.

[26] E. Perelman et al. Using Simpoint for Accurate and Efficient Simulation. In *SIGMETRICS*, 2003.

[27] M. K. Qureshi et al. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *MICRO*, 2009.

[28] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *ISCA*, 2009.

[29] L. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. Technical Report DCS–TR–675, Dept. of Comp. Science, Rutgers Univ., Sept. 2010, Revised Mar. 2011.

[30] S. Rixner et al. Memory Access Scheduling. In *ISCA*, 2000.

[31] K. Sudan et al. Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement. In *ASPLOS*, 2010.

[32] D. Wang et al. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News*, 33(4), 2005.

[33] M. Wu and W. Zwaenepoel. eNVy: a Non-Volatile, Main Memory Storage System. In *ASPLOS*, 1994.

[34] Xilinx. Spartan-6 FPGA Memory Controller User Guide. 2010. http://www.xilinx.com/support/documentation/user_guides/ug388.pdf.

[35] B.-D. Yang et al. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *ISCAS*, 2007.

[36] L. Zhang et al. The Impulse Memory Controller. *IEEE Transactions on Computers, Special Issue on Advances in High-Performance Memory Systems*, November 2001.

[37] W. Zhang and T. Li. Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In *PACT*, 2009.

[38] P. Zhou et al. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *ISCA*, 2009.

[39] Y. Zhou, P. Chen, and K. Li. The Multi-Queue Replacement Algorithm for Second-Level Buffer Caches. In *USENIX*, 2001.