Real-Time Scheduling for Phase Change Main Memory Systems

Miao Zhou, Santiago Bock, Alexandre P. Ferreira, Bruce Childers, Rami Melhem, Daniel Mossé
Department of Computer Science, University of Pittsburgh
{*miaozhou, sab104, apf75, childers, melhem, mosse*}@*cs.pitt.edu*

***Abstract*—Multi-core processors are effective for reducing energy consumption in computer systems, since modern multi-core chips allow for power management of individual cores. However, multiple cores impose higher demand on the memory subsystem, which is extremely power hungry. In addition to the small steps towards managing power in DRAMs, Phase-Change Memory (PCM) has emerged as a low-power alternative that is especially helpful for energy-aware embedded real-time systems. However, there are three drawbacks to PCM: its high latency, high energy consumption when writing, and low endurance. In real-time systems, the impact of PCM's high access latency is of special interest, as it has a negative effect on the number of deadlines that are met by the system.**

In this paper, we examine the memory subsystem and add a real-time scheduler for prioritizing requests at the bottleneck resource, the PCM controller. Adding support for *external priorities*, we use rate monotonic (RM) and earliest deadline first (EDF) prioritization at the PCM and show that it does reduce the number of deadline misses, but not sufficiently. We examine two additional schemes for prioritizing PCM requests (*critical read boosting* and *read over write*). We show that the scheduler of the PCM controller has a significant influence on the percentage of missed deadlines: critical read boosting and read over write can reduce the percentage of missed deadlines by 80% in the best case with negligible energy overhead.

I. Introduction

Modern processors have been dodging the frequency wall by increasing the number of cores on a chip. They have also lowered energy/power demand by implementing either dynamic voltage and frequency scaling (DVFS) for sets of cores or by allowing systems to turn cores on and off independently. This capability is particularly important in real-time systems because most real-time systems are designed to function in the worst-case, and therefore, these systems are typically highly overprovisioned. Research continues in processor power management, and in the last decade, some of the research has been incorporated into general-purpose and real-time operating systems (OS).

Especially in multi-core systems, the memory subsystem has become a major consumer of power, which has fueled research in alternative memory technologies. Phase-Change Memory (PCM) [1–3] is proposed as either a replacement for DRAM or in addition to DRAM (using a smaller DRAM as a cache for the larger PCM). PCM is desirable because it is non-volatile, scales better than DRAM, has low power for reads and very low idle power. However, PCM has unique challenges: it is wearable [4–7], and it is slower and consumes more energy on writes than DRAM.

The memory hierarchy in a multi-core processor typically has on-chip L1 and L2 caches. When data is not cached, requests go from the CPU to the main memory through a memory controller. Traditionally, the memory controller has a simple design, processing requests in a First-In-First-Out (FIFO) manner. However, with the advent of modern memory controllers, more advanced techniques, such as prioritizing memory requests, have been suggested [8–12]. These techniques are typically for fast DRAM and have small influence on application throughput.

The system we consider in this paper consists of a small DRAM cache and a large PCM main memory [13–15]. This architecture is amenable to embedded systems, as they are becoming increasingly more power-constrained. Samsung has already announced the integration of DRAM and PCM devices into a multichip package (MCP) in smartphones.

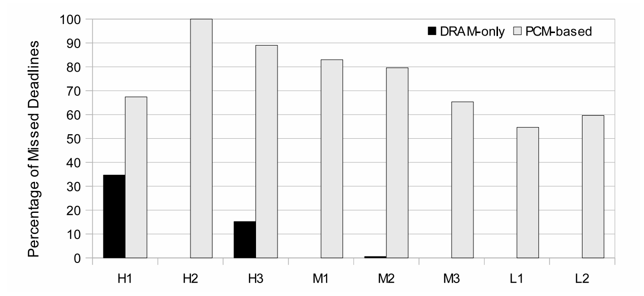

Figure 1. Percentage of missed deadlines under different memory systems

The characteristics of PCM, however, make the adoption of PCM in real-time systems challenging. In particular, PCM's high access latency can negatively impact the number of deadlines that are met. Figure 1 shows the percentage of missed deadlines for eight task sets in two different memory systems, with a laxity factor of 1.001.[1] Both memory systems apply FIFO to schedule memory requests. It is clearly shown that a PCM-based memory system causes more tasks to miss the deadline. For instance, task set H2 meets all the deadlines in a DRAM-only system, while it misses all deadlines in a PCM-based system. This means that a naive scheduling policy is not sufficient to make PCM a feasible choice for real-time embedded systems.

This paper prioritizes the bottleneck resource–PCM operations–in a hybrid memory subsystem. We examine the

[1]More experimental details are described in Section V. The architectural details for the PCM-based system are presented in Section II-B.

influence of a typical real-time scheduler (earliest-deadline-first, EDF, or rate monotonic, RM) when prioritizing requests that arrive at the PCM controller. We note that the prioritization based on EDF or RM is slightly more complex than FIFO because the priority is based on the deadline and/or period of the task. This priority cannot be derived solely from the request type or the request arrival time. There has to be communication between the OS and the memory controller, to propogate the priority/deadline information from the software to the hardware layer. This information must be taken into account by the PCM controller.

We also adopt two ways to reorder requests at the PCM controller, in addition to external priorities. First, *critical read boosting* (CRB) gives high priority to *critical* requests (i.e., a read that is stalling the processor) over *non-critical* requests (i.e., a prefetching read). CRB is similar to *critical word first* [16]. However, CRB also reorders requests across tasks executing in a multi-core system. Second, *read over write* reduces the amount of time that reads must wait for high latency writes to complete. Clearly, these memory prioritization techniques can also be applied to a DRAM-only system. However, the special characteristics of PCM make these techniques more valuable in a PCM-based memory system. As we show in Section V, the impact of our techniques is considerably larger in a PCM-based system than in a DRAM-only one.

Our results show that latency and deadline misses are significantly reduced with three techniques: (a) external priorities, (b) critical read boosting and (c) read over write. In the best case, critical read boosting, combined with read over write, reduces the percentage of missed deadlines by up to 80% with negligible energy overhead.

In this paper, we make the following contributions:

- We study the prioritization of memory operations in the context of multi-core real-time systems with PCM and provide evidence that PCM request scheduling has a significant impact on the number of missed deadlines.
- We present a successful solution to the problem of scheduling PCM memory operations through existing real-time scheduling policies in the memory controller. We also provide solutions that apply architectural techniques to prioritize PCM memory requests.
- We sketch the OS and hardware support required to enable real-time scheduling.
- We provide an extensive and accurate evaluation of the solutions and analyze their impacts on the number of missed deadlines, energy consumption and latency.

The remainder of this paper is organized as follows. Section II provides the background knowledge. Section III describes our memory access scheduler. Section IV presents the OS and hardware support required to implement priorities in the memory controller. Section V gives the evaluation of our techniques, Section VI describes related work, and Section VII concludes the paper.

II. Background

A. Phase Change Memory

PCM is a type of non-volatile memory that stores information by changing the state of a chalcogenide material. By applying electrical currents of different intensities and durations, the physical state or phase of the material can be changed from amorphous to crystalline and vice versa. Since the configuration of atoms in the material is different for each stat, the energy required to change the stored value is much higher. This property implies that the state persists for longer periods of time without the need of power hungry refreshes. PCM consumes very little power when idle, which makes it an ideal fit for embedded real-time systems.

PCM does, however, require more energy for writes. In addition, writes to PCM are about 5-15 times slower than to DRAM, while reads are 2-3 times slower. PCM has the ability to write to individual memory locations, unlike NAND Flash that must erase and write memory block.

Although these properties make PCM a good replacement for DRAM, one of the main drawbacks of PCM is its limited endurance such that only about 10^7 or 10^8 writes can be performed. This problem has been addressed by many researchers [4–7,15].

B. Architecture Overview

Our memory request scheduler is based on a hybrid memory architecture called Phase Change Main Memory Architecture (PMMA) [14]. PMMA (depicted in Figure 2) is composed of three components: the Memory Manager (MM), the Acceleration and Endurance Buffer (AEB) and the PCM main memory. The AEB is a DRAM cache for the larger PCM; we refer to it as DRAM or AEB interchangeably. Since memory access latency is much larger for PCM, having a cache that filters most of the accesses to PCM improves performance considerably. The cache also helps extend PCM lifetime by reducing PCM write traffic.

The MM manages the internal operation of PMMA and handles the flow of information between the CPU, AEB and PCM. The MM has two internal memory controllers (DRAM controller and PCM controller), which control the operation of the AEB and PCM. The MM also contains an In-Flight

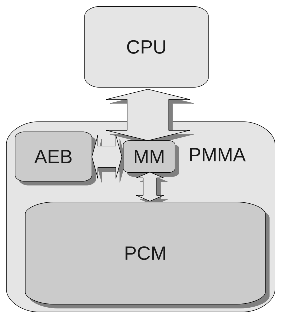

Figure 2. PMMA architecture

Buffer (IFB), which is a high-speed memory (SRAM) that temporarily stores data as it is transferred between the CPU, AEB and PCM. Additionally, the MM contains a tag array for data that is stored in the AEB and a request buffer with the state of pending transactions to the memory subsystem.

To improve locality and bandwidth utilization, the controllers ensure that the size of each block of data stored in the memory subsystem (referred to as a page) is larger than a cache line. Requests for cache lines arriving at the MM will be transformed into requests for pages; this prefetching is common in memory architectures.

PMMA supports the critical word first (CWF) optimization, which is common in many cache hierarchies. When a request for a cache line arrives, the MM fetches the entire page containing the cache line from PCM. Without CWF, if the requested cache line is not the first line of the page, the processor must wait for all previous cache lines of the page to be transferred. With CWF, the MM first fetches the requested cache line and then the rest of the cache lines of the page. Thus, the requested cache line is sent back to the processor earlier, reducing execution time.

III. Scheduling at the PCM Controller

This section discusses the enhanced memory scheduling policies, including critical read boosting, external priorities and read over write.

The PCM controller is responsible for interfacing with the actual PCM device. When a page is allocated to the AEB, the MM issues a request to the PCM controller to read the page from the device. Rather than always servicing a complete page from PCM, our PCM controller allows for breaking pages into smaller units (called sub-requests), and services these sub-requests according to their priorities [14]. The size of each sub-request is set to the largest burst size of the PCM device to maximize bandwidth utilization.

A. External Priorities

Real-time systems often give high priority to some tasks to avoid missing deadlines. In some architectures, however, simply assigning a higher priority to a task might not be enough because other resources are not prioritized. For example, in a multi-core system with shared memory, a task that has a core assigned exclusively to it might still miss its deadline due to possible memory contentions.

We use *external priorities* to manage PCM requests of different tasks. These priorities are assigned externally by the user or OS and must be passed to the memory subsystem. The PCM controller uses the priorities to schedule the pending PCM requests so that requests from high priority tasks are executed first. Currently, the only information used by the controller is the type and arrival time of requests.

External priorities can be assigned in many ways. Real-time scheduling techniques that are used for CPU scheduling can be used to prioritize requests at the PCM controller. For example, priorities can be set according to the *earliest deadline first* (EDF) scheduling algorithm, which assigns the highest priority to the task with the earliest deadline. Alternatively, a *rate monotonic* (RM) scheduling discipline would derive the priorities from the task periods, with the highest priority assigned to the task with the shortest period.

B. Critical Read Boosting

In a multi-core system, memory requests from different cores are intertwined together to form a single request stream. Two requests from different cores can arrive approximately at the same time to the PCM controller. The simple case is when both these requests can be satisfied by the fast AEB. We consider below the more important case: when data for both requests resides in the PCM. Since PMMA brings large pages to the AEB to support prefetching and applies CWF, it is possible that the transfer of the critical cache line of a request is delayed by the non-critical cache lines of other requests. This has a negative impact on both the number of deadline misses and latency because the core must stall waiting for the non-critical reads to finish. The purpose of prefetching cache lines to the AEB is to accelerate future memory requests to adjacent data locations. However, these cache lines are not immediately needed and should not interfere with critical memory requests. Note that when we apply EDF and RM, this is still the case, given that the priority of the subpages is the same for an entire request.

To avoid the impact of prefetching on the latency of other memory requests, we adopt *critical read boosting* (CRB): requests for critical cache lines are given higher priority than requests for non-critical cache lines, even if the cache lines belong to different pages. Figure 3 shows the scheduling decisions for both CRB and CWF. The first core requests cache line *A2*, which is part of a page containing 3 other cache lines (*A1*, *A3* and *A4*). The second core requests the third cache line *B3* of another page that also includes cache lines *B1*, *B2* and *B4*. Assuming request *A* arrived first, critical cache line *A2* is scheduled first. When CRB is applied, instead of scheduling the other cache lines of *A*, the priority of cache line *B3* is *boosted* so that it is scheduled before the non-critical cache lines of *A*. Finally, the non-critical cache lines of *A* and *B* are scheduled.

In contrast to CRB, CWF only prioritizes critical cache line within each page, and does not reorder requests across page boundaries. In this example, CWF does not schedule

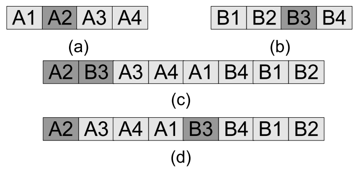

Figure 3. Comparison of critical read boosting and critical word first. a) First request with critical cache line A2. b) Second request with critical cache line B3. c) Queue after both requests arrive for critical read boosting. d) Queue after both requests arrive for critical word first.

critical cache line *B3* before non-critical ones *A3*, *A4* and *A1* (see Figure 3(D)), and tasks might be delayed.

Note that a sub-request must not necessarily be the same size as the cache line used for critical read boosting. If the sub-request is larger than the cache line, then the entire sub-request to which the cache line belongs is considered critical. If the cache line is larger than the sub-request, then multiple sub-requests would be considered critical.

C. Read over Write

Generally, writes are not in the critical path of execution because they can be buffered and performed while a core continues to do useful work (i.e., cores do not stall waiting for completion of writes). However, the latency of critical reads can be influenced by writes when reads and writes compete for a resource, such as the PCM. If reads often wait for writes to complete, tasks will be delayed and might miss the deadlines. As mentioned before, PCM writes have a much higher latency than reads. Therefore, reads are more likely to wait for writes in PCM-based memories.

Read over write is applied to mitigate the effects of slow writes on read latency. When using read over write, PCM read requests are always prioritized over PCM write requests. Note that all write requests to the PCM are due to evictions of dirty data from the AEB. As such, write requests do not contain a critical cache line and are scheduled always after pending (critical and non-critical) reads.

IV. Operating System and Hardware Support

Unlike a FIFO scheduler that requires only local information, and therefore, can schedule PCM operations solely with information at the PCM controller, any scheduling discipline based on information about the tasks (e.g., the deadline or the period) needs a mechanism to communicate the priority from the operating system (OS) to the memory controller. We assume that the OS is aware of the scheduling policy and that it has access to the information required to calculate each task's priority. Alternatively, the priorities can be assigned by the user, in which case the same mechanism for passing priorities is required.

Each core of a modern processor has a so-called program status word (PSW), which is a register with information about the active task. The PSW can be augmented to include a new field with the priority of the current task. This field is set by the OS when a task is scheduled or its priority changes, according to the scheduling policy and the information about the task. The OS uses a privileged instruction to write to the PSW.

When the processor sends a request to the MM, the PSW of the core that generated the request is looked up and the value of the priority field is sent along with the request to the MM, which ultimately propagates it to the PCM controller. The PCM controller can then use this priority internally to reorder its queue of sub-requests.

To achieve this reordering, the protocol between the memory controller and the MM has to be changed to accommodate the new field. This could be done by adding a new type of control word (command) before the regular request is issued. This new control word is sent on the control bus and requires no response. To avoid having to issue this command for every memory operation, the priority itself could be embedded as part of the request. In this case, there has to be enough available space in the original command to accommodate the new priority field.

Alternatively, if the overhead of issuing the new command for each memory operation is too high and there is no space available in the original command to embed the priority, a scheme based on memory ranges can be used instead. In this case, the communication protocol between the memory controller and the MM would be augmented with a new command that defines ranges of physical memory that share the same priority. This command would be issued by the memory controller for each memory range that belongs to the scheduled task. The MM can keep a list of these ranges and their associated priority. Every time a memory request comes, the MM can look up the corresponding entry in the list and determine the priority of the request. The list would have to be the same size as the number of entries in the TLB of each core, since this would cover all memory that is currently being used by all cores.

V. Evaluation

A. Methodology

We use an in-house simulator to evaluate the performance and energy impact of memory scheduling policies. The simulator uses accurate timing and energy models. The input to the simulator is one or more memory reference traces, obtained by running benchmarks on Simics, which is a well-accepted cycle-accurate simulator for multi-core architectures [17]. The memory trace contains, for each memory request by the CPU, the time stamp (assuming zero memory latency, that is, counting only CPU cycles to execute the task and L1/L2 cache latency), the type of request (read vs. write) and the physical address of the memory reference.

Our simulator supports both DRAM-only and PCM-based memory system (i.e., PMMA). It has a memory activity generator that processes the memory trace and initiates memory requests to the memory hierarchy. The simulator models the MM, DRAM/PCM controllers and devices.

The simulator schedules all shared resources (e.g., bus transations) and accounts for latency due to resource contention. Both DRAM and PCM controllers have internal queues of finite size (parameterizable). A queue-full signal halts the sender until the queue can accept new requests.

B. Experimental Setup

The Simics configuration used to generate the traces has four 3GHz x86 processors, each with a L1 and a L2 cache. The L1 I-cache and D-cache are 4-way, 32 KBytes with 64

Parameter	PCM	DRAM
Bus Size (Bits)	8	8
Bus Cycle Time (ns)	16.7	3
Read Latency (ns)	40	15
Write Latency (ns)	160	15
Read Bus Speed (MHz)	66	333
Write Bus Speed (MHz)	33	333
Idle Current (mA)	1	7
Read Current (mA)	10	160
Write Current (mA)	70	160
Vdd (V)	1.8	1.8

Table I
PARAMETERS USED IN THE SIMULATION

Benchmark	Request rate	AEB miss rate	PCM request rate	Type
susan.smoothing	432	13.08	56	Low
dijkstra	716	9.13	65	Low
gsm.encode	657	10.29	68	Low
gsm.decode	841	13.64	115	Low
fft	1804	7.36	133	Low
fft.inverse	2645	6.84	181	Medium
patricia	1573	18.11	285	Medium
crc32	790	42.53	336	Medium
susan.edges	1816	19.12	347	Medium
jpeg.encode	2258	16.64	376	Medium
sha	1056	50.7	535	Medium
jpeg.decode	2494	25.91	646	High
pgp.verify	3607	18.41	664	High
adpcm.decode	1634	49.83	814	High
adpcm.encode	2380	34.97	832	High
susan.corners	2869	33.29	955	High
pgp.sign	4771	22.95	1095	High
stringsearch	3762	35.77	1346	High

Table II
MEMORY REQUEST RATE (IN REQUESTS PER MILLION CYCLES), AEB MISS RATE (%) AND PCM REQUEST RATE (IN REQUESTS PER MILLION CYCLES) OF SEVERAL MIBENCH BENCHMARKS

bytes cache lines and 1 cycle hit latency. The L2 is an 8-way, 128 KBytes unified instruction and data cache with 64 bytes cache lines and 6 cycles hit latency. The traces were collected until the benchmarks finish execution.

The simulation parameters are shown in Table I. The DRAM-only memory system is configured with a 333MHZ 64bit DDR2 bus with 16 DIMMs. A PCM-based memory system consists of the PCM memory which is arranged as a 16GBytes logical space, and a DRAM cache of 4MBytes. We evaluated PCM-based memory system with different DRAM cache sizes (4, 8 and 16MBytes), and the results are similar. For the rest of the paper, we only show results for PCM-based system with a 4MBytes DRAM cache. PCM devices are interconnected to the MM via a single 66MHz 128-bit DDR2 bus. We configured the PCM controller to use a queue size of 1024 requests. The PCM timing and power model come from a Samsung prototype [2].

To evaluate our techniques, we use a set of tasks with different memory request rates and different AEB miss rates derived from the MiBench benchmark suite [18]. Table II shows the detailed information of several benchmarks.

We categorized the benchmarks into three types, according to their PCM request rate (high, medium and low). We use this metric because it provides a good estimate of the pressure on the memory subsystem and how much a task's execution time is influenced by the performance of PCM. Based on these properties, we created eight different *task sets*, which are sets of four tasks to be run at the same time in different cores. Task sets are classified according to the PCM request rate of their tasks, as shown in Table III. While high (low) rate task sets contain only high (low) rate benchmarks, medium rate task sets can contain medium rate benchmarks, a combination of high and low rate benchmarks or a combination of the three types.

Our simulator is configured to run all four tasks of a set concurrently. Each core repeats the execution of one of the tasks until the task with the longest period has been executed for 10 times. The period (deadline) of each task is computed as the execution time of the task when it is run alone in one core multiplied by a *laxity factor*. We ran our experiments with laxity factors between 1 and 1.5 and steps of 0.001.

C. *Missed Deadlines*

This section evaluates how different real-time scheduling policies influence the fraction of deadline misses. We show results for six policies: First-In-First-Out (FIFO), Rate Monotonic (RM), Earliest Deadline First (EDF), Critical Read Boosting (CRB), Read over Write (RoW), and a combination of Critical Read Boosting and Read over Write (CRB+RoW). Our evaluation shows that different combinations of real-time scheduling policies (e.g., RM+CRB+RoW)

Task set name	Type	Benchmark 1	Benchmark 2	Benchmark 3	Benchmark 4
H1	High	stringsearch	pgp.sign	pgp.verify	susan.corners
H2	High	susan.corners	jpeg.decode	adpcm.encode	adpcm.decode
H3	High	pgp.sign	pgp.verify	susan.corners	jpeg.decode
M1	Medium	susan.corners	jpeg.decode	dijkstra	susan.smoothing
M2	Medium	susan.corners	susan.edges	fft.inverse	susan.smoothing
M3	Medium	fft.inverse	susan.edges	jpeg.encode	sha
L1	Low	dijkstra	susan.smoothing	gsm.decode	gsm.encode
L2	Low	dijkstra	fft	susan.smoothing	gsm.decode

Table III
LIST OF TASK SETS AND THEIR CORRESPONDING BENCHMARKS.

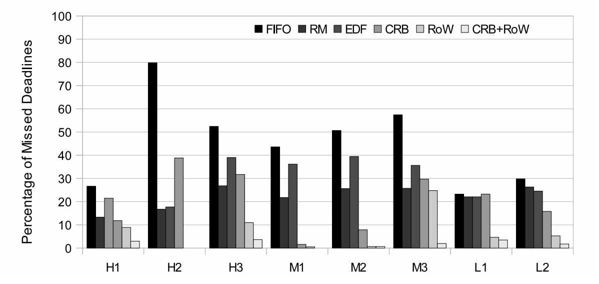

Figure 4. Percentage of missed deadlines of a PCM-based memory system

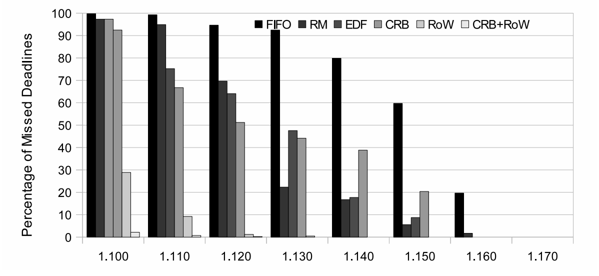

Figure 5. Percentage of missed deadlines of task set H2 in a PCM-based memory system with different laxity factors

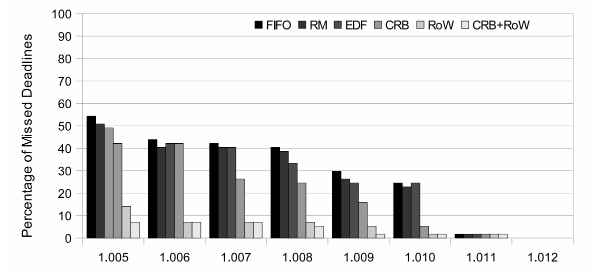

Figure 6. Percentage of missed deadlines of task set L2 in a PCM-based memory system with different laxity factors

produce similar results. Thus, we present results for only one combination of policies (CRB+RoW).

Figure 4 shows the effect of the real-time scheduling policies on the percentage of deadline misses for the PCM-based system. For each task set, the fraction of missed deadlines for FIFO is greater than any of the other policies (RM, EDF, CRB and RoW), as expected. Note that the laxity factor used to obtain the values shown in this figure is different for each of the eight task sets.

RM, EDF, CRB and RoW reduce the number of missed deadlines considerably for task sets with high and medium PCM request rates. For example, for task set H2 and a laxity factor of 1.140, FIFO causes 80% of the tasks to miss their deadlines. In contrast, RM and EDF miss only 20% of the deadlines and CRB+RoW finishes all tasks on time.

For task sets with low PCM request rates (e.g., L1 and L2), RM/EDF can only improve the fraction of missed deadlines slightly. On the other hand, CRB/RoW can reduce

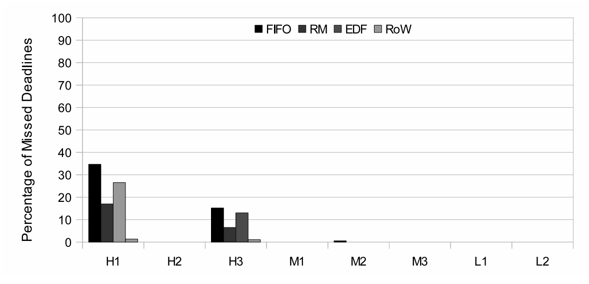

Figure 7. Percentage of missed deadlines of a DRAM-only system (laxity factor of 1.001)

the deadline miss percentage of these task sets significantly. In other words, CRB/RoW is able to benefit a broader range of task sets than RM/EDF. This happens because CRB and RoW improve performance of all tasks, not just those with higher priorities. CRB prioritizes critical read requests of all tasks and RoW prioritizes read requests (including critical and prefetching read requests) of all tasks. This effect is more pronounced in task sets with high and medium rate benchmarks, because the fraction of time spent waiting for PCM memory operations is higher, which provides more opportunities for improvement.

Figures 5 and 6 show the percentage of missed deadlines as a function of laxity factor for H2 and L2, respectively. CRB and RoW scheduling policies have impact over a wider range of laxity factors for H2 than for L2. This is also due to the larger amount of time that the processor must wait for PCM operations when executing high request rate tasks.

As mentioned before, we expect the impact of our techniques to be very small for DRAM-only systems. This assumption is well supported by the results in Figure 7, which shows the effect of different real-time scheduling policies in a DRAM-only system. CRB is not included in these results because there is no prefetching from a lower level of the memory hierarchy in a DRAM-only system and CRB requires the distinction between critical and prefetching requests. For all but two task sets, the difference between policies is negligible. In addition, the impact on the other two task sets is also relatively small. Although the graph only shows the percentage of missed deadlines for a laxity factor of 1.001, we note that choosing a smaller laxity factor (1, for example), causes the system to miss all deadlines irrespective of the scheduling policy. This helps further demonstrate our assumption about DRAM-only systems.

D. Energy

Figure 8 shows the energy consumption of different scheduling policies in a PCM-based memory system normalized to that of FIFO in a DRAM-only system. In a PCM-based memory system, the energy overhead of our scheduling policies is within 5% of the energy consumption of the FIFO policy. The energy overhead of RM, EDF, CRB and RoW are higher for task sets with high and

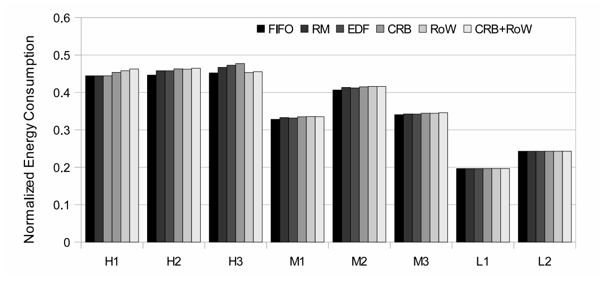

Figure 8. Energy consumption of different scheduling policies in a PCM-based memory system normalized to that of FIFO in a DRAM-only system

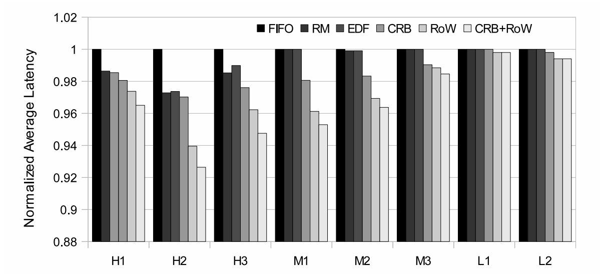

Figure 9. Normalized average latency of eight task sets in a PCM-based memory system with laxity factor of 1.5

medium PCM request rates. This happens because these policies cause more tasks to complete, consuming more energy. More importantly, the energy consumption of a PCM-based system is only 20% to 48% of that of a DRAM-only system. As shown above, for a PCM-based system, our scheduling policies can significantly reduce the number of missed deadlines, similar to that of a DRAM-only system. However, our approach achieves this with up 80% energy reduction, compared to a DRAM-only system.

E. Latency

We showed that our scheduling policies reduce the number of missed deadlines significantly for overloaded systems. Our techniques are also beneficial even when the system is underutilized. Figure 9 shows the average latency of all tasks normalized to FIFO. We choose a laxity factor of 1.5 to make sure all tasks meet their deadlines. While task sets with low PCM request rates are not helped by CRB and RoW, CRB+RoW reduces the average latency by up to 7% for task sets with high/medium PCM request rates.

VI. Related Work

Memory access scheduling has been extensively studied in stream processors [8,19–21], general-purpose single-core processors [22–24] and general-purpose multi-core processors [9–11,25–29]. In this paper, we focus on reducing the number of deadline misses in real-time systems.

Ipek *et al.* propose a self-optimizing memory controller based on the principles of reinforcement learning [11]. Kim *et al.* propose a new technique for memory access scheduling in multi-core systems with several memory controllers [12]. Our work differs from these studies in the memory architecture we are targeting. Whereas they focus only on common DRAM architectures, our work centers on a hybrid DRAM-PCM system.

Nesbit *et al.* propose a fair memory scheduler for high-performance chip multiprocessors, based on scheduling algorithms for network fair queuing [9]. Mutlu *et al.* provide a fair memory scheduler that attempts to equalize the slowdown experienced by each thread due to interference caused by other threads when accessing memory [26]. In [10], they present another memory controller that divides requests in several batches, each of which can be optimized independently, hence improving fairness. All these approaches provide fairness and can be used to provide quality of service. However, none of these studies consider the effects of the scheduling decisions on real-time systems.

Burchard *et al.* designed a real-time streaming memory controller for shared memory interconnect-centric systems [21]. This study considers quality of service guarantees for accessing memory. However, this controller was specifically designed for streaming applications. The study did not evaluate the performance of general-purpose multi-core systems.

Rosen *et al.* study several optimizations for predictable memory controllers in chip multiprocessors [27]. They try to optimize access to the bus, which connects several cores with several private memories. The work focuses mainly on predictable memory controllers for real time systems, not the reduction of missed deadlines or latency.

Hennessy and Patterson develope critical word first to reduce the cache miss penalty [16]. The cache requests the missed word first from the memory and sends it to the CPU as soon as it arrives. The CPU continues execution while filling the rest of the words in the cache line. However, this scheme does not take into account multi-core systems where a non-critical cache line requested by one core might delay a critical line requested by another core. In such case, tasks might miss the deadline due to the delayed critical requests.

Qureshi *et al.* try to allow reads to proceed without waiting for writes to finish by enforcing write cancellation and write pausing [30]. This work focuses on reducing latency by preempting the PCM writes that are already in execution when another PCM read request arrives. However, it does not study real-time systems or the impact of these techniques on missed deadlines. Moreover, we showed that prioritizing reads over writes is enough to achieve low miss deadlines for PCM-based memory systems, and cancelling or pausing the on-going PCM write operations is not necessary.

VII. Conclusion

Due to the active researches on processor power management, DRAM has become a major power consumer for embedded and real-time systems. To combat power consumption, PCM has emerged as an alternative main memory architecture; however, PCM is slow, and thus architects use it together with a small DRAM cache. Given the effort to increase application throughput by increasing the number of cores, the memory subsystem has become a major performance bottleneck.

We showed in this paper that prioritizing requests at the bottleneck resource, namely PCM, using task information, increases the chance that tasks meet their deadlines. We discussed how to carry out the communication between the OS and the hardware to make priority information feasible at the memory controller. We also suggest reordering PCM requests based on criticality of the requests to reduce latency and the number of missed deadlines. This is, to the best of our knowledge, the first attempt to insert real-time priorities in the memory controller. Perhaps this is because DRAM-only memory systems do not profit from these prioritization techniques, as DRAM is fast enough not to be a bottleneck in embedded real-time systems.

References

[1] Kwang-Jin Lee et al., “A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput,” *Solid-State Circuits, IEEE Journal of*, vol. 43, 2008.

[2] Kang et al, “A 0.1 μm 1.8V 256Mb 66MHz Synchronous Burst PRAM,” in *ISSCC ’06*, 2006.

[3] F. Pellizzer et al., “A 90nm phase change memory technology for stand-alone non-volatile memory applications,” in *Symp. on VLSI Technology*, 2006.

[4] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “A durable and energy efficient main memory using phase change memory technology,” in *ISCA ’09*, 2009.

[5] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, “Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling,” in *MICRO 42*, 2009.

[6] S. Cho and H. Lee, “Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance,” in *MICRO 42*, 2009.

[7] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, “Increasing pcm main memory lifetime,” in *DATE ’10*, 2010.

[8] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” in *ISCA ’00*, 2000.

[9] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, “Fair queuing memory systems,” in *MICRO 39*, 2006.

[10] O. Mutlu and T. Moscibroda, “Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems,” in *ISCA ’08*, 2008.

[11] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana, “Self-optimizing memory controllers: A reinforcement learning approach,” in *ISCA ’08*, 2008.

[12] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, “Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers,” in *HPCA ’10*, 2010.

[13] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting phase change memory as a scalable dram alternative,” in *ISCA ’09*, 2009.

[14] A. P. Ferreira, B. Childers, R. Melhem, D. Mosse, and M. Yousif, “Using pcm in next-generation embedded space applications,” in *RTAS ’10*, 2010.

[15] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology,” in *ISCA ’09*, 2009.

[16] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[17] Magnusson, Peter S. et al., “Simics: A full system simulation platform,” *Computer*, vol. 35, 2002.

[18] Guthaus, M. R. et al., “Mibench: A free, commercially representative embedded benchmark suite,” in *WWC ’01*, 2001.

[19] McKee, Sally A. et al., “Dynamic access ordering for streamed computations,” *IEEE Trans. Comput.*, vol. 49, 2000.

[20] T.-C. Lin, K.-B. Lee, and C.-W. Jen, “Quality-aware memory controller for multimedia platform soc,” in *SIPS ’03*, 2003.

[21] A. Burchard, E. Hekstra-Nowacka, and A. Chauhan, “A real-time streaming memory controller,” in *DATE ’05*, 2005.

[22] S. Rixner, “Memory controller optimizations for web servers,” in *MICRO 37*, 2004.

[23] I. Hur and C. Lin, “Adaptive history-based memory schedulers,” in *MICRO 37*, 2004.

[24] J. Shao and B. T. Davis, “A burst scheduling access reordering mechanism,” in *HPCA ’07*, 2007.

[25] C. Macian, S. Dharmapurikar, and J. Lockwood, “Beyond performance: secure and fair memory management for multiple systems on a chip,” in *FPT ’03*, 2003.

[26] O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in *MICRO 40*, 2007.

[27] Rosen, Jakob et al., “Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip,” in *RTSS ’07*, 2007.

[28] T. Moscibroda and O. Mutlu, “Distributed order scheduling and its application to multi-core dram controllers,” in *PODC ’08*, 2008.

[29] C. J. Lee, O. Mutlu, V. Narasiman, and Y. Patt, “Prefetch-aware dram controllers,” in *MICRO 41*, 2008.

[30] M. Qureshi, M. Franceschini, and L. Lastras-Montano, “Improving read performance of phase change memories via write cancellation and write pausing,” in *HPCA ’10*, 2010.