An Efficient Parallel Scheduling Scheme on Multi-partition PCM Architecture

Wen Zhou, Dan Feng, Yu Hua, Jingning Liu[*] , Fangting Huang, Yu Chen
Wuhan National Lab for Optoelectronics, School of Computer Science
Huazhong University of Science and Technology
Wuhan 430074, China
{zhouwen, dfeng, csyhua, jnliu, huangfangting, chenyu0713}@hust.edu.cn

ABSTRACT

Phase Change Memory (PCM) is an emerging non-volatile memory with the salient features of large-scale, high-speed, low-power and radiation resistance. It hence becomes an ideal candidate for the next-generation storage media of main memory. However, PCM suffers from inefficient I/O performance due to long write latency. Recent studies propose a *multi-partition* (or *multi-subarray*) architecture within each bank to enhance internal parallelism. However, conventional scheduling schemes fail to exploit the advantage of multiple partitions and incur inefficient bank utilization. In this paper, we propose a Write Priority overlap Read (WPoR) scheduling scheme which preferentially serves for a write request in one partition and allows other partitions to perform as many read requests as possible within this partition's program duration. Experimental results demonstrate that WPoR reduces the write latency by 24.7% (on average) compared with state-of-the-art scheduling algorithms. Meanwhile, the IPC indicator of WPoR scheduling increases respectively 6%, 7% and 26% (on average) compared with Read Priority, Write Pausing and Write Cancellation schemes.

CCS Concepts

•Information systems → Phase change memory; •Hardware → Memory and dense storage;

Keywords

PCM; multiple partitions; parallel scheduling

1. INTRODUCTION

Currently, applications have higher requirements on the processor and memory system. Over the past decades, the rapid development of multi-core technology has improved the parallelism and the performance of the processors, which allows many threads or applications to run simultaneously [19]. However, traditional memory systems fail to meet the needs of fast I/O operations and large capacity. The performance gap between computation (e.g., the multi-core processors) and storage (e.g., DRAM) causes the performance degradation in the entire system. The advent of 20-nanometer semiconductor technology has been challenged by power and scalability of the refresh operation in Dynamic Random Access Memory (DRAM) medium. For example, Lefurgy [9] pointed out 40% of the energy consumption by the main memory in the IBM eServer servers. Non-volatile memory technology can replace DRAM and build a large-capacity and energy-efficient storage system, such as Phase Change Memory (PCM), Magnetic Random Access Memory (MRAM) and Resistive Random Access Memory (ReRAM). Among them, PCM is the most promising candidate for building main-memory systems [8, 15, 20].

Compared with DRAM technology, PCM shows asymmetrical read/write latency. The latency of write operation is $4 \sim 8$ times longer than that of read operation, which incurs severe bank conflicts. The long write latency is more likely to block incoming memory accesses to the same bank. Bank conflicts, coupled with the write latency, significantly reduces memory bandwidth utilization and can cause cores to stall, leading to lower system performance [7]. In order to address this problem, recent studies [18] propose an accurate-and-tight power management technique for the PCM system, which exploits current difference between writing bit 0 and bit 1 so as to leverage the surplus unused power budget to serve more requests within a bank. This technique has been applied into the practical PCM chips [2, 12], which develop a multi-partition architecture within each bank. The new chips show the advanced feature that a read operation is possible to be served in one partition while write in another partition. Thus, it provides a new way to enhance system performance by exploring partition-level parallelism. Compared with banks that are the smallest independent structure, the parallelism among partitions has more restrictions. For example, two or more write operations are not allowed to be simultaneously performed in different partitions within a bank for limited power constraint. Therefore the partition-level parallelism is weaker than the bank-level one.

Although the new architecture provides advanced features, existing scheduling schemes show inefficiency on multi-partition architecture. For example, the First Come First Serve (FCFS) scheduling serves for requests according to their arrival orders, while the Read Priority scheme preferentially serves for read requests. Hence, the array program durations (i.e., the internal execution phase of the write requests) are not fully utilized. Based on the above observation, we propose an efficient scheduling scheme, called WPoR. By changing the serving order of read/write requests, WPoR allows more read requests to be served within the program duration of the write requests. Finally, WPoR

[*]Jingning Liu is the corresponding author.

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA
© 2016 ACM. ISBN 978-1-4503-4185-1/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934583.2934610

scheduling reduces the whole runtime and saves energy consumption simultaneously.

In this paper, the main contributions can be summarized as follows.

First, we introduce a novel multi-partition PCM model to address the poor I/O performance problem existing in traditional PCM architectures. After that, we explore and exploit the feasible parallelism within PCM banks to improve PCM performance.

Second, to improve PCM bank utilization, we propose WPoR scheduling scheme which exploits the restricted parallelism among the partitions, overlapping the program duration of write request to serve for read requests that have different partition addresses. In addition, we propose the effective address mapping scheme to enhance partition-level parallelism under various applications.

Finally, simulation experiments are conducted on Gem5 simulator [1], and we evaluate WPoR and other scheduling schemes by the indicators in terms of the average read/write latency, throughput and IPC. Experimental results show that WPoR performs better than existing schemes in terms of memory throughput and system IPC.

The rest of this paper is organized as follows. Section 2 introduces the background and motivation for the WPoR research. Section 3 gives detailed design of WPoR scheme and Section 4 shows the extensive evaluation of WPoR. Section 5 summarizes the related work of PCM-based scheduling algorithms. Finally, in Section 6, we conclude the paper.

2. BACKGROUND AND MOTIVATION

PCM is a non-volatile memory device and uses two states of $Ge_2Sb_2Te_5$ (GST) phase-change material to store data. GST material is used to quickly heat and quench the glass or hold it in its crystallization temperature range for some time, thus making it high-resistance amorphous state or switching it to a low-resistance crystalline state. Under the two states, GST material has different resistance values. Data make use of this characteristic for storage. Reading the content of PCM cells requires low and short current to sense the material resistance value. Therefore, both energy consumption and latency of the write operation are higher than those of read operation.

To migrate the poor write performance, scientists propose a multi-partition architecture within each bank [2, 12]. In this section, we analyze the timing constraints of real chips to help us exploit the partition-level parallelism.

Firstly, we refer to a unified PCM timing model proposed in [11] to simplify the operating procedure of read and write requests. The read and write requests are implemented by a series of bank commands, including Preactive (*PRE*), Active (*ACT*), Read (*RD*) and write (*WR*). The combination of PRE and ACT commands is to fetch the data of that memory line from the PCM array to row buffer. The RD command obtains data from the given row buffer, while the WR command transmits the data to the row buffer and then write it to PCM cell. Therefore, the read request is executed by the PRE-ACT-RD command sequence, and the write request is satisfied by the PRE-ACT-WR command sequence. For write request, data are not actually written to the memory array immediately after the write commands are issued into the PCM bank, but will delay for a long time. This stage is known as *array program*. Usually, a bank does not receive external commands when the memory array is in array program state. Otherwise, the on-going operation will be terminated. Fortunately, the multi-partition technique separates the read and write operations in different partitions. Hence, a read and a write requests can be performed within a bank concurrently, as shown in Figure 1 (a). R2 request with partition address being PA1 can be served within the array program duration of W1 request with partition address (PA0). Moreover, for current constraints, a read and a write requests are not allowed to perform concurrently within a partition. As shown in Figure 1 (b), R3(PA0) will be served after W1(PA0) completes, which incurs a partition conflict. The partition conflict will obstruct the following requests and decrease bank utilization.

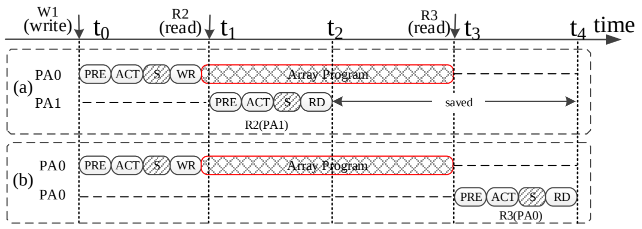

Figure 1: The timing constraint of partition-level parallelism.

Although the multi-partition PCM provides partition-level parallelism, existing scheduling schemes fail to explore this advantage. We illustrate the effect of request scheduling between write and read requests on runtime with an example. As shown in Figure 2, there are five requests served on the bank with five scheduling schemes. The partition address of each request is referred on the bracket.

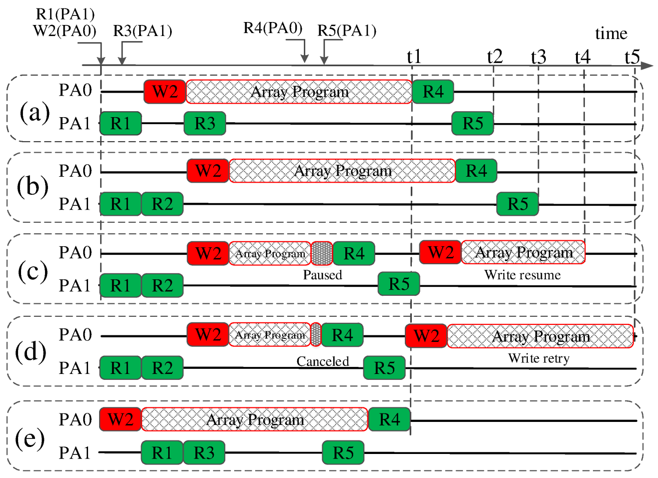

Figure 2: The timing diagram showing the runtime of four read requests and one write request under five scheduling algorithms: (a) FCFS scheduling; (b) Read Priority Scheduling; (c) Read Priority + Write Pausing; (d) Read Priority + Write Cancellation; (e) Overlapped Scheduling.

We analyze this stream on five systems with different management schemes respectively. Figure 2 (a) shows the timeline of a system that uses the FCFS scheduling. Such a system serves for requests according to their arrival orders. We observe that only R3 can be overlapped served with W2 request. Compared with the single-partition system, the runtime of FCFS scheduling scheme with multi-partition system reduces tR (i.e., a read latency). Figure 2 (b) exhibits the timing diagram of Read Priority scheduling. To reduce read latency, R1 and R3 are performed before W2 request. As R4 is conflicted with W2 request, R4 is blocked until W2 completes. The Read Priority scheduling does not reduce runtime. Considering inevitable partition conflicts existed in the FCFS and Read Priority scheduling schemes, the write pausing and write cancellation commands are explored to further reduce the waiting time for incoming read requests. The Write Pausing scheme suspends W2 instantly and serves for R4 and R5 requests. After the pending read requests are completed, a resume command is required to re-active W2, as shown in Figure 2 (c). The Write Cancellation terminates W2 instantly and serves for the pending read requests. As shown in Figure 2 (d), after R4 and R5 are completed, an integrated write command is executed to re-serve W2. Write Pausing and Write Cancellation schemes introduce extra overheads and thus result in longer runtime. The above four algorithms fail to leverage the advantages of the parallel operations in the multi-partition architecture, and lead to relatively long runtime. In this paper, we propose an overlapped scheduling scheme to exploit partition parallelism of overlapping write requests to serve read requests among different partitions. As shown in Figure 2 (e), after re-ordering the pending requests, R1, R3 and R5 requests can serve with W2 in parallel, and the runtime saves $3\times tR$. The proposed overlapped scheduling requires shorter runtime and saves more energy compared with existing scheduling algorithms.

In the real-world environments, scheduling algorithms also need to consider dynamic request queues and read/write partition conflicts. Therefore, we propose WPoR scheme to exploit the advantages of multi-partition parallelism. More details about WPoR are discussed in the next section.

3. DESIGN AND IMPLEMENTATION

Deploying multi-partition PCM in a practical system proves to be a promising solution to bridge the gap between the fast-running multi-core processor and the long-access-latency secondary storage devices. However, as is well known, PCM naturally has long write latency. Hence, to balance read and write latency, practical systems usually employ a small-sized DRAM as PCM's buffer. The overall architecture of such system is shown in Figure 3. The memory controller has a read queue and a write queue, which stores the pending read and write requests, respectively. In general, the read requests are derived from the processor directly, while the write requests are generated by DRAM buffer to evict the dirty data. To achieve high system performance, a scheduler is responsible to schedule these requests efficiently. For example, once the read queue is full, the memory requests derived from the processor should be blocked.

The existing scheduling schemes, designed for DRAM or conventional PCM devices, adopt a serial scheduling pattern and thus show poor efficiency on multi-partition PCM devices. In this section, we design an efficient parallel scheduling scheme, called WPoR, which allows more read requests to be overlapped served with the ongoing write requests in parallel. In what follows, we illustrate the procedure of WPoR scheduling in detail.

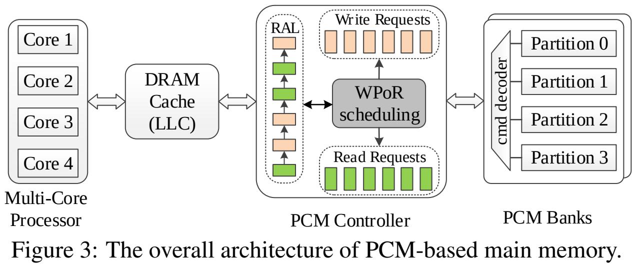

Figure 3: The overall architecture of PCM-based main memory.

3.1 The Scheduling of WPoR Scheme

The goal of WPoR scheduling is to allow more requests to serve on PCM bank parallelly, thus achieving shorter response time and higher I/O throughput. By using WPoR, for memory-sparse applications, the controller could activate relevant banks and respond the pending requests one by one. For memory-intensive applications, since there are too many requests in the controller queues, WPoR reordering the serving sequence of the requests to exploit the potential partition-level parallelism.

To achieve this goal, WPoR employs several strategies. Generally, a write request has higher priority than a read request in WPoR because write operation has longer latency and obviously should be responded earlier. After the timing commands of a write request are issued, the relevant bank turns into the array program state. During the write operation stage, WPoR could serve for remainder read requests whose addresses are in different partitions with the on-going write operation. The ideal situation is that all the read requests in read queue have no conflicts with the write requests in write queue. In this situation, all read requests are able to be processed with write requests in parallel, and they are served according to their arrival orders.

However, in most cases, there are two or more read/write requests with same partition address in the controller queue. Hence, WPoR needs to tune the serving order of read requests to prevent such potential partition conflicts. Obviously, the conflicting read requests should be maintained in controller queue and will be scheduled after the on-gong write request completes. Given that a read request may be blocked too long, we leverage a timeout threshold for the pending read requests and WPoR will serve for the overtime read requests preferentially.

Since WPoR scheduling changes the serving order of read and write requests, the memory systems suffer from potential read-after-write hazard. To prevent data contention, we design an address comparison circuit which consists of a Requested Address List (RAL) to store the metadata of the pending requests. The RAL entries are sorted according to their arriving orders. Each entry contains a request number, read/write type and physical address. Once a write request is picked out, WPoR removes relevant entry from RAL and compares its physical address with the earlier arrived read requests'. If their physical addresses are identical, the write request will be blocked. Moreover, when a read request is selected to be served, WPoR compares its address with earlier arrived write requests. If they are the same, the read data could be obtained from the write request directly. Above all, WPoR is able to serve for read/write requests in parallel, avoiding data contention.

In summary, the complete procedure of WPoR scheduling can be summarized as follows. When PCM bank turns into idle state, the memory controller first serves for the timeout read requests from read queue. Second, PCM activates one of the earliest arrived write requests. Third, during the array program stage of the write request, PCM circularly performs the read requests that have different partition addresses with the on-going write request. In the meantime, the address comparer checks those read requests to avoid data contention. Finally, as the write request completes, WPoR enters a new round and PCM could serve for the next write request instantly.

3.2 Effective Address Mapping Scheme

Previous researches [6, 17] have shown that spatial locality is widely existed in various applications. Sequential Requests can be served from the row buffer with lower latency than accessing the PCM array. Hence, exploiting the row-buffer locality with row-interleaved address mapping to enhance PCM bank's performance is meaningful. However, with the increasing trend of worker processes, a recent study [11] demonstrates that the memory requests show weak locality in the practical systems. Hence, the block-interleaved address mapping achieves higher performance than row-interleaved mapping, especially under applications with random access patterns. Because the read and write requests are distributed in different partitions, which reduces partition conflicts and is good for WPoR scheduling. Therefore, we adopt block-interleaved mapping scheme in our system.

4. PERFORMANCE EVALUATION

4.1 Experiment Setup

We evaluate the WPoR algorithm on Gem5 simulator [1] that is a full-system simulation tool developed by the University of Michigan. Gem5 is an open-source architecture simulator that is capable of emulating various instruction sets, processors and memory models. We extend multi-partition PCM memory model on the Gem5 simulator, and implement several request scheduling algorithms in the memory controller, including FCFS, Read Priority, Write Cancellation and Write Pausing. Table 1 shows the configuration parameters of our simulation system. The processor contains an 8-core CMP, 32KB L1 Cache and 2MB L2 Cache. Main memory contains a 32MB pure-write DRAM buffer that is organized as a private cache for PCM. Both read and write latencies of DRAM are 50ns. The 16GB PCM-based main memory contains 2 channel and 8 banks within each channel. A bank consists of 4 partitions. The read and write latencies of PCM are 250ns and 2us, respectively [14].

Table 1: The configuration of the simulation system

Processor	8 cores, in-order CMP, 3.2GHz, 32KB L1 instruction/data cache, 2MB L2 cache
DRAM Cache	32MB, 8-way, 64B linesize, 50 ns access latency, writeback policy
Memory Controller	16 GB PCM, 400MHz, Offchipbus, 128-entry request queues, 2 channels, 8 banks and 4 partitions
PCM latency	250ns read latency, 2us write latency

Table 2: The characteristic of the mixed benchmarks

Name	Description	Read PKI	Write PKI	WSS (MB)
astar	8 copies of 473.astar	8.0	4.7	208
bwaves	8 copies of 410.bwaves	36.2	34.1	3794.4
dealII	8 copies of 447.dealII	0.2	0.1	117.6
gamess	8 copies of 416.gamess	1.6	0.03	4.8
gcc	8 copies of 403.gcc	4.7	1.3	527.2
gobmk	8 copies of 445.gobmk	45.2	42.7	132
leslie3d	8 copies of 437.leslie3d	20.5	19.4	601.6
mcf	8 copies of 429.mcf	3.9	2.0	5446.4
namd	8 copies of 444.namd	0.53	0.49	81.6
perlbench	8 copies of 400.perlbench	1.5	0.8	51.2
soplex	8 copies of 450.soplex	3.4	0.7	217.6
zeusmp	8 copies of 434.zeusmp	57.3	54.2	2160.8
gromacs	8 copies of 435.gromacs	1.5	0.23	68.8
hmmer	8 copies of 456.hmmer	1.07	1.03	65.6
libquantum	8 copies of 462.libquantum	7.8	7.4	261.6
wrf	8 copies of 481.wrf	12.1	11.4	1308

We use 16 benchmarks from the SPEC CPU2006 suite which is widely used for evaluating the performance of the main memory [4, 10]: *astar*, *bwaves*, *dealII*, *games*, *gcc*, *gobmk*, *leslie3d*, *mcf*, *namd*, *perlbench*, *soplex*, *zeusmp*, *gromacs*, *hmmer*, *libquantum* and *wrf*. These benchmarks are chosen due to their various densities of memory access. In order to measure the maximum service capability of various scheduling algorithms, we conduct a pressure test on main memory and run 8 copies of these benchmarks in the full system. Each copy runs in an independent CPU core. Therefore, the mixed benchmarks have higher memory access frequency. We adopt block-interleaved technique in address mapping ensuring high partition-level parallelism. Table 2 shows the amount of memory access, read (RPKI) and write (WPKI) requests per 1000 instructions out of the 32MB DRAM cache. The Working Set Size (WSS) is an estimate of how much memory is actively used by an application. The result from Table 2 exhibits that the smaller value of WSS is, the lesser write requests are served by PCM because many dirty data have been absorbed by DRAM cache. To accurately evaluate the performance of the main memory, we analyze the results after running 5×10^{15} ticks.

4.2 Experiment Results

We evaluate the performance of FCFS, Read Priority, Write Cancellation, Write Pausing and WPoR algorithms under the same configuration. We use average response time of read/write requests, memory throughput and system IPC as our indicators.

4.2.1 The Average Response Time of Read Requests

Figure 4 shows the average response time of read requests under five scheduling algorithms. The results exhibit that the FCFS scheme has the longest read latency than others and the average value reaches up to 1263 ns (on harmonic mean). The main reason is that the read requests are not allowed to be processed until all the early arrived write requests complete, even if the bank has idle partition to serve read requests during write requests being served. Especially in the write-intensive applications, such as *bwaves*, *gobmk*, *zeusmp*, *libquantum* and *wrf*, the WPKIs reach or approach to 10 and the average read latencies exceed 2us. The Read Priority scheme that gives higher priority to read requests has an obvious effect on reducing read latency. The average read latency under Read Priority scheme decreases to 720 ns which is 43% lower than that of FCFS algorithm. Moreover, the Write Cancellation and Write Pausing algorithms that terminate the on-going write requests are able to achieve lower response latencies for read requests, and the average latencies decrease to 445 ns and 500 ns, respectively. It is worth noting that the Write Pausing has higher read latency than that of the Write Cancellation algorithm due to the long suspending latency brought by pausing operation, during which the bank prohibits all the external commands. In contrast to previous algorithms, the read latency of WPoR scheduling is slightly lower than that of Read Priority and the average value achieves 694 ns.

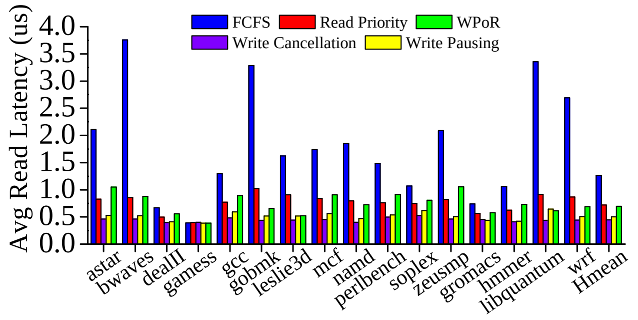

Figure 4: Average response time of read requests under five scheduling algorithms.

4.2.2 Average Response Time of Write Requests

Figure 5 shows the average response time of write requests under five scheduling algorithms. Although the Write Cancellation and Write Pausing schemes have the lower read latency, they terminate the on-going write requests and incur expensive write penalty. The results from Figure 5 show that the write latencies of Write Cancellation and Write Pausing increase to 94.9us and 30.7us, which are longer than Read Priority's (i.e., 27.9 us). The write latency of WPoR algorithm is slightly higher than that of FCFS scheduling (i.e., 18.2us), reaching 23.6 us. Because WPoR scheduling suffers from more numbers of memory requests in the controller queue.

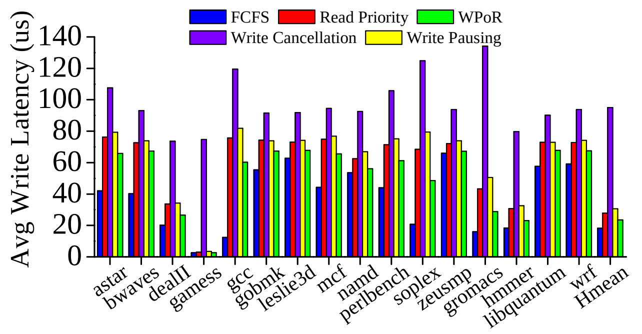

Figure 5: Average response time of write requests under five scheduling algorithms.

4.2.3 *Memory Throughput and System IPC*

Since the values of the memory throughput and system IPC span a large range under various benchmarks, we use normalized indicators to evaluate the various scheduling schemes. We use FCFS as a baseline approach. Figure 6 shows the normalized throughput of scheduling algorithms under 16 benchmarks. The results exhibit the performance improvements on WPoR scheduling are 6%, 7% and 26% higher than Read Priority, Write Pausing and Write Cancellation, respectively. In this multi-partition architecture, bank conflicts rarely occur and read requests wait for a short time to be served in most case, so that Write Cancellation and Write Pausing cannot obtain better performance by reducing read latency. Instead, pausing and cancellation overheads have many negative effects on performance. Also, the experiments demonstrate exploiting scheduling schemes to avoid partition conflicts achieve better performance than using advanced commands to address conflicts.

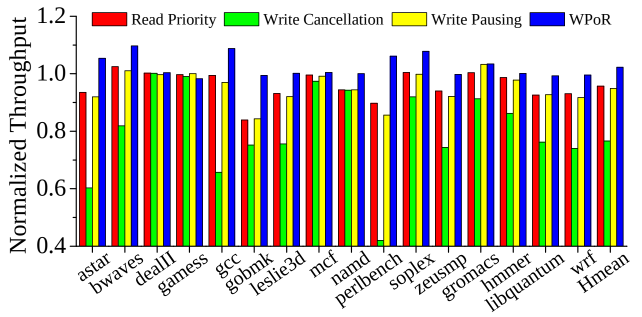

Figure 6: The memory throughput of Read Priority, Write Cancellation, Write Pausing and WPoR normalized to FCFS (Baseline).

Figure 7 shows the normalized IPC of scheduling schemes. Since system IPC is linearly associated with memory throughput, the results show the similar feature as memory throughput. Although WPoR incurs slightly higher read latency compared with Write Cancellation and Write Pausing schemes, it shows the highest efficiency for memory utilization with hybrid read and write requests. Hence WPoR show higher IPC performance than other policies.

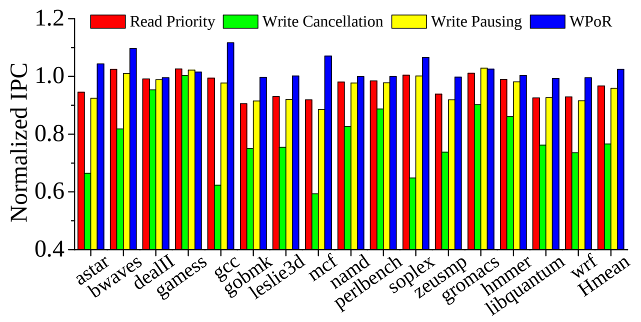

Figure 7: The System IPC of Read Priority, Write Cancellation, Write Pausing and WPoR normalized to FCFS (Baseline).

4.2.4 *Hardware Overhead*

In this subsection, we estimate the hardware overhead of the WPoR algorithm. Given that K represents the length of read/write queue and S denotes the size of memory request, the read/write queues takes up K $\times$ S. The data of the read and write requests are stored on SRAM in memory controller. The RAL consists of K $\times$ C entries, where C denotes the size of each entry, including address, type and request number. To reduce address-comparison latency, we store RAL entries on registers in memory controller.

According to our experimental configuration, the read and write queue consists of 128 entries, respectively. Hence, the spatial overhead of request queue reaches 2 $\times$ 128 $\times$ 256B= 64KBytes, while that of RAL occupies 2 $\times$ 128 $\times$ 12B= 3KBytes (each item includes 4Byte) register resources. In short, the WPoR scheduling algorithm consumes affordable hardware overhead.

In summary, without optimizing the memory scheduling, the FCFS scheme shows the worse performance than WPoR. Especially in read latency, FCFS is 2 times longer than WPoR. Read Priority that first serves for read requests exacerbates partition-level parallelism, incurring longer read/write latencies and lower system IPC than those of WPoR. Write Pausing and Write Cancellation sacrifice their write performance to obtain better read latencies, but the extra write penalty overheads decrease the entire system performance. Finally, our experimental results demonstrate that we take both read and write requests into account to achieve the significant performance improvements. Based on this observation, our WPoR scheduling preferentially serves for write request and overlaps the array program duration to serve for read requests, and achieves higher system IPC than the state-of-the-art scheduling algorithms.

5. RELATED WORK

Development on multi-partition (multi-subarray) architecture. Recently, many studies exploit the existence of partition (subarray) within each bank mitigating the effect of bank conflicts. Kim et al. [7] propose a subarray-level parallelism (SALP) mechanism to explore timing constraints among subarrays within each DRAM bank, which overlaps the latency of accesses to different subarrays. Inspired by the multiple subarrays technology in DRAM, Yue et al. [18] propose a new power allocation scheme to explore the existence of subarray in PCM system. The scheme leverages subarray-level parallelism to enable a bank to serve for a write and multiple reads in parallel without violating power constraints. Micron and Samsung propose multi-partition based chips [2] [12], which support parallel read and write operations for users. Therefore, the multi-partition architecture has significant performance advantage. However, to the best of our knowledge, the partition-level based scheduling has been rarely touched by existing work. Our work is the first work for PCM scheduling on multi-partition architecture, which leverages the unique features of PCM.

Scheduling policies for PCM. PCM has long write latency, incurring poor I/O performance. There are several scheduling policies trying to address this problem. Read Priority scheduling [3] is widely used in asymmetric NVM devices, which gives a high priority to read requests and thus reduces waiting time for pending read requests. In order to reduce waiting time of incoming read requests which is obstructed by on-going write requests, Qureshi et al. [14] propose write cancellation and write pausing scheduling policies, which terminate the on-going write request and response pending read requests instantly. Although these schemes further reduce the read delay, they cause nontrivial pausing and cancelling overheads, which reduce service capacity and consume more energy. In comparison with these schemes, our WPoR prevents conflict and conceals long write latency by intelligent scheduling. In addition, there are several more complex algorithms for the asymmetric NVM devices, such as Flash I/O scheduler (FIOS) [13], Round Robin (RR) [16] and Completely Fair Queuing (CFQ) [5] algorithms, which can be applied to the PCM device. These policies achieve an OS-level fairness by allocating a time slice for each request or thread. These OS-level scheduling algorithms are orthogonal to hardware-level scheduling algorithms and can co-operate with our WPoR to optimize the I/O performance in different level. Existing scheduling schemes [7] designed for multi-subarray DRAM show inefficiency on PCM since long write latency is not hidden. The work proposes the partition-level scheduling to exploiting the parallelism among partitions. The scheme is more suitable for applying in multi-partition PCM-based main memory.

6. CONCLUSION

Phase Change Memory (PCM) is a promising candidate for building main memory systems. One drawback of PCM is its long write latency which incurs severe bank conflicts and poor I/O performance. In this paper, we introduce a multi-partition PCM architecture and propose WPoR scheduling scheme for this architecture to address these problems. WPoR allows more read requests to be served along with write requests in parallel. In the meantime, we have evaluated the WPoR algorithm on Gem5 simulator. Experimental results demonstrate that WPoR scheduling performs better than the state-of-the-art scheduling policies, such as Read Priority, Write Cancellation and Write Pausing, on memory throughput and system IPC.

Acknowledgment

This work was supported by the National High-tech R & D Program of China (863 Program) No. 2015AA016701, No. 2015AA015301, No. 2013AA013203; NSFC No. 61303046, No. 61402189, No. 61173043; State Key Laboratory of Computer Architecture under Grant CARCH201505. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

7. REFERENCES

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011.

[2] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, et al. A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth. In *ISSCC*, 2012.

[3] C. Dirik and B. Jacob. The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. In *ACM SIGARCH Computer Architecture News*, 2009.

[4] K. Ganesan, J. Jo, and L. K. John. Synthesizing memory-level parallelism aware miniature clones for spec cpu2006 and implantbench workloads. In *ISPASS*, 2010.

[5] S. Hui, Z. Rui, C. Jin, L. Lei, W. Fei, and X. C. Sheng. Analysis of the File System and Block IO Scheduler for SSD in Performance and Energy Consumption. In *Asia-Pacific Services Computing Conference*, 2011.

[6] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez. Balancing dram locality and parallelism in shared memory cmp systems. In *HPCA*, 2012.

[7] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *ISCA*, 2012.

[8] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. *ISCA*, 2009.

[9] C. Lefurgy, K. Rajamani, F. L. R. III, W. M. Felter, M. Kistler, and T. W. Keller. Energy Management for Commercial Servers. *IEEE Computer*, 2003.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[11] J. Meza, J. Li, and O. Mutlu. Evaluating Row Buffer Locality in Future Non-Volatile Main Memories. *Carnegie Mellon University Technical report*, 2012.

[12] Micron Inc. Product Brief LPDDR2-PCM and Mobile LPDDR2 121-Ball MCP. http://caxapa.ru/thumbs/441272/LPDDR2-PCM_Br.pdf.

[13] S. Park and K. Shen. FIOS: a fair, efficient flash I/O scheduler. In *FAST*, 2012.

[14] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montao. Improving read performance of Phase Change Memories via Write Cancellation and Write Pausing. In *HPCA*, 2010.

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

[16] M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round Robin. *Computer Communication Review*, 1995.

[17] K.-C. Wong, K.-S. Leung, and M.-H. Wong. Effect of spatial locality on an evolutionary algorithm for multimodal optimization. In *Applications of evolutionary computation*. 2010.

[18] J. Yue and Y. Zhu. Exploiting subarrays inside a bank to improve phase change memory performance. In *DATE*, 2013.

[19] D. Zhan, H. Jiang, and S. C. Seth. Stem: Spatiotemporal management of capacity for intra-core last level caches. In *43rd MICRO*, 2010.

[20] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.