Integration and Boost of a Read-Modify-Write Module in Phase Change Memory System

Hyokeun Lee, *Student Member, IEEE*, Moonsoo Kim, Hyunchul Kim, Hyun Kim, *Member, IEEE*, and Hyuk-Jae Lee, *Member, IEEE*

Abstract—Phase-change memory (PCM) is a non-volatile memory device with favorable characteristics such as persistence, byte-addressability, and lower latency when compared to flash memory. However, it comprises memory cells that have limited lifetime and higher access latency than DRAM. The row buffer size of a PCM is preferred to be larger than 128B to fill the latency gap between two memories and to reduce the metadata overhead incurred by wear leveling. As the cache line size in a general-purpose processor is 64B, a read-modify-write (RMW) module is required to be placed between the processor and the PCM, which in turn induces a performance degradation. To reduce such an overhead and enhance the reliability of a device, this paper presents a new RMW architecture. The proposed model introduces a DRAM cache in the RMW module, which minimizes redundant read operations for write operations by pre-fetching the entire transaction unit instead of merely caching the 64B requested data. Furthermore, a typeless merge operation is performed with the proposed cache by gathering multiple commands accessing consecutive addresses, irrespective of whether they are *READ* or *WRITE*. Simulation results indicate that the proposed method enhances the speed by 3.2 times and the reliability by 49 percent as compared to the baseline model.

Index Terms—Phase-change memory, non-volatile memory, read-modify-write, command merging

1 Introduction

Owing to its characteristics of non-volatility, byte addressability, and relatively low latency, a phase-change memory (PCM) device is gaining attention as the next-generation non-volatile memory (NVM) device, along with ReRAM and STT-MRAM [1], [2], [3]. These characteristics allow new non-volatile memories to replace an existing main memory or storage by adding a flexible new memory layer for the current computer architecture hierarchy. In recent years, 3D-XPoint has been proposed as a gap filler between memory and conventional storage systems [4], [5], and software-defined persistent memory (SDPM) has been proposed to utilize non-volatile memory as fast storage or expanded main memory [6], [7], [8]. Additionally, with the growing demand for high-performance server systems in the future, an increasing amount of main memory is further desired, such as an in-memory database. Furthermore, an in-memory database needs to let application data be persistent in storage, which incurs substantial performance overheads [9], [10]. A PCM is an adequate candidate for the purposes mentioned above because it is a low latency non-volatile memory device. Therefore, exploiting these technologies is highly crucial to attaining a high-speed and large-capacity memory system in the future.

Despite its characteristics of non-volatility and high endurance, which make it superior to NAND flash memory, PCM is not ready to be prevalently commercialized owing to two well-known problems. The first problem is a large latency for a write operation that requires no less than 1000ns, which is much slower than that of DRAM [11], [12], [13], [14]. The large latency can be compensated for by the improved throughput that is achieved by increasing the size of a memory transaction (i.e., row buffer size) rather than using the conventional 64B as in [2]. In [2], it is pointed out that a 512B row buffer offers a good trade-off between delay and energy consumption. Furthermore, doubling bit width has been widely adopted for enhancing the bandwidth in high bandwidth memory (HBM) or flash memory devices [15], [16], [17], [18], [19].

The second problem that prevents a PCM from commercialization is the limited write endurance of 1E8 per cell. As a result, wear leveling is required to maintain uniformity in write access across memory spaces [20], [21], [22], [23], [24]. A wear leveling algorithm, such as start gap or security refresh, divides the entire memory space into several sub-regions and maintains metadata like *gap pointers* and *swap keys* for each sub-region to prevent it from *Repeated Address Attack* [22], [23]. Therefore, the size of this sub-region should be large to reduce resource overheads while maintaining the reasonable degree of write uniformity. As a result, a large number of the wear leveling algorithms for a PCM

- *H. Lee, M. Kim, and H. J. Lee are with the Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea. E-mail: {hklee, kimms213, hyuk-jae-lee}@capp.snu.ac.kr.*
- *H. Kim is with SK Hynix Inc., Icheon, Gyeonggi-do, South Korea. E-mail: hyunchul3.kim@sk.com.*
- *H. Kim is with the Department of Electrical and Information Engineering, Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea. E-mail: hyunkim@seoultech.ac.kr.*

Manuscript received 18 Nov. 2018; revised 17 July 2019; accepted 28 July 2019. Date of publication 8 Aug. 2019; date of current version 5 Nov. 2019. (Corresponding author: Hyun Kim.)
Recommended for acceptance by Z. Shao.
Digital Object Identifier no. 10.1109/TC.2019.2933826

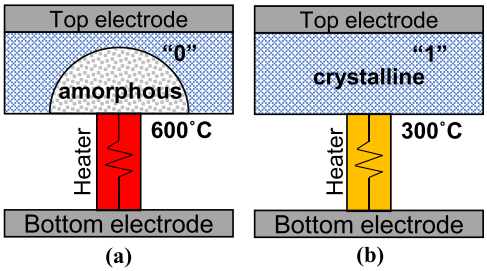

Fig. 1. Cell structure of phase-change memory. (a) amorphous state, (b) crystalline state.

select the size of the sub-region larger than 128B instead of 64B [18], [19].

The conventional general-purpose processor typically adopts a 64B cache line, which is also the basic access unit of a main memory system for each transaction. Therefore, a read-modify-write (RMW) operation is essential to handle the size gap between a PCM (i.e., larger than 128B) and a cache line size (i.e., 64B) [25]. The redundant read operation in RMW for write access causes performance degradation. As no efficient RMW scheme for a PCM has been studied so far, the RMW schemes proposed for other devices can be applied for PCM access. In [26], a coalescing operation gathers write requests in RMW for a GPU, which is highly effective because write access patterns of most GPU applications are sequential, and hence suitable for coalescing write requests. However, adopting this method for a general-purpose system is ineffective because the access patterns of general-purpose applications are generally read-dominant and randomly accessed. In [27], RMW with command combining has been proposed for efficient error correction coding (ECC), but it assumes that the ECC codeword length (i.e., eventually decided transaction unit) is equal to the cache line, so it does not consider the case for a larger row buffer size of the memory device. For design simplicity, experiments in previous PCM-related researches assume that the cache line size and the row buffer size are identical without considering the non-symmetric case [2], [21], [22], [28], [29]. The RMW should be carefully optimized in the design of a practical PCM-based system because the RMW for a PCM may cause a performance degradation owing to the redundant read operations and the frequent read operations to a specific cell cause read disturbance errors as discussed in [30].

In this research, a novel RMW method is proposed to minimize the physical burden on PCM devices by using a DRAM cache between the PCM and processor. Previous researches have been conducted to improve the performance of PCM-based main memory system. The proposed method uses the DRAM cache for data pre-fetching so that it can further reduce the number of read-modify-write operations. As a result, the use of cache reduces the direct access to PCM devices, and thus, enhances the read reliability. For further optimization, multiple commands are merged into a single memory command regardless of the types of operations, generating consecutively aligned addresses constituting the memory row buffer size. Consequently, for a PCM with a row buffer size of 512B and a 2 MB cache between the PCM and the processor, the combination of two contributions increases the speed by nearly 3.2 times when compared to the baseline system. The read reliability is also improved by 49 percent on average. In summary, the major contributions of this paper are as follows:

- An RMW integrated into a PCM-based system is presented to provide a practical place-holder reference for upcoming new processor architecture. This is the first attempt to adopt RMW in the design of a PCM-based memory system.
- A method of integrating DRAM cache and RMW module in a PCM-based system is introduced. The proposed method is cost effective because it utilizes the existing DRAM cache, which is quite common for a PCM-based system as shown in prior work.
- The proposed architecture is additionally optimized with a novel and simple *merge* operation regardless of the types of commands, which maximizes the performance and reliability of the system.

The rest of this paper is organized as follows. Section 2 presents the background and the baseline system. In Section 3, the baseline RMW is described in details. In Sections 4 and 5, both DRAM cache-based RMW and *typeless merge operation* are proposed, respectively. The evaluation of the speed and error reliability with various parameters is given in Section 6. Finally, the last section presents the conclusions.

2 Background and Motivation

2.1 Introduction to Phase-Change Memory

Phase-change memory is a type of resistive memory device that exploits two inter-changeable phases (i.e., the crystalline state and the amorphous state) of a Ge2Sb2Te5 (i.e., GST). The amorphous state of a device is obtained by heating the bottom electrode of the device (see Fig. 1) until the temperature exceeds 600 degrees Celsius, which increases the resistance of the device. On the other hand, the crystalline state, or SET logic level, is reachable by supplying a temperature of 300 degrees Celsius to the material [12]. Therefore, this process makes the write operation of the PCM slower than reading the cell resistance by supplying the sensing voltage [31].

Because PCM has lower latency than NAND flash memory, it is considered as one of the most next-generation memory technologies for high-performance data servers, such as in-memory databases [9], [10]. It can also endure more writes than NAND flash memory, typically a maximum of 1E7 1E8 writes per cell which makes the device more attractive than other non-volatile memory elements.

However, PCM still has a relatively higher latency than DRAM, and hence the throughput of the system is enhanced by increasing the transaction size of PCM [2], or by performing read operations under write operations [32]. Additionally, PCM has lower reliability than DRAM. This is because the frequent read operations to a specific cell would lead to read disturbances [30], so both aspects need to be considered simultaneously while designing the system.

2.2 Previous Research

There are some previous works for enhancing the performance of PCM-based main memory system. Basically, incorporating a DRAM cache into a PCM-based memory system

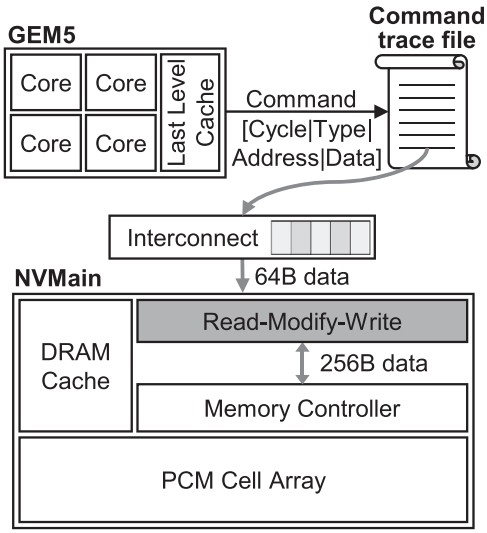

Fig. 2. Baseline system environment.

are made to build a hybrid memory system and enhance the lifetime and performance by utilizing the DRAM as a carrier of dirty pages in [31] and [33], respectively. In [11], the SET operation of a write command, which has a longer latency than RESET operation, is issued ahead from the memory controller when the memory becomes idle, and thereby only RESET operation is required for that write command because a PCM adopts differential write scheme in the device. In [32], it shows that the latency of write operation can be hidden by adopting reads-under-write method with the introduction of write-status hold register (WSHR) in a memory controller and four row buffers for read commands (based on the LPDDR2 standard). It leverages the stabilization time of write operations on PCM device to hide the latency by accessing multiple row buffers in a non-blocking way. In [12], a non-blocking PCM bank design is proposed in which the roles of a sense-amplifier and a write driver in a bank are separated. Therefore, they can work concurrently through write-precedence scheduling policy. Additionally, a device design that enables multiple activations at tile-level, called Fine-Grained NVM (*FgNVM*), is proposed to exploit parallelism for hiding more latencies in [34]. Thus, both a write command and a read command can be processed in more fine-grained ways than bank interleaving to improve the performance of the memory system. Similarly, devices in a rank of a DIMM is designed to work independently to accommodate fine-grained commands for higher command processing throughput [35].

Both the mentioned works generally assume that the cache line size of a processor and the transaction unit of a PCM system are equal. Although [32] mentions the problem that the speed performance degrades with increasing write units, it does not put further effort on read-modify-write processes (see details in Section 2.5). Therefore, a compatible RMW is desirable for matching these two different sizes.

2.3 Baseline System and Its Configurations

Fig. 2 depicts the baseline system where a PCM is a part of the heterogeneous main memory system that includes a DRAM cache as in previous research [21], [22], [28], [29], [31], [33]. An FTL-like controller is equipped with the currently available NVDIMM [36], [37]. The RMW can be included as a part of the memory controller or as an individual entity separated from the controller [26], [27], [38], [39]. In this paper, the RMW module is separated from the controller for implementation simplicity and functional extension in the future.

In this paper, two simulators are used to build the baseline system. First, an out-of-order 4-core processor with L2 last level cache (LLC) having 64B line size is configured in *gem5* based on ARM Cortex A53 as shown in Table 1. Subsequently, the PCM system with memory controller, banks, and rank models is built with *NVMain* [40]. The RMW module is additionally implemented for cycle-accurate simulation of RMW operations. For the configuration of timing parameters in NVMain, tRCD (row-to-column delay) is set to 50ns and tWP (write pulse time) is set to 1000ns.

Trace-based simulation is conducted to reduce the simulation time. Trace files of workloads are extracted from gem5 in system emulation mode, and the detailed information of the workloads is described in the next subsection. Each line of the trace file is extracted from the output path of the LLC. The trace line consists of a CPU cycle, type, command address, and command data (both read and write), and each trace line is recognized as one memory command. Due to the behavioral differences between gem5 and NVMain even when the same timing parameters are chosen, the command cannot flow into the NVMain at the cycle inscribed in the

TABLE 1
Simulation Configurations

gem5 (trace extractor)			
Processor	Caches		
	L1 Cache		L2 Cache
Out-of-Order, 4-core, 2 GHz	I-Cache 2-way set associative, 64 KB	D-Cache 4-way set associative, 64 KB	Shared last level cache 16-way set associative, 1 MB, 64B cache line
NVMain			
Interconnect	32-entry asynchronous FIFO		
Memory information	400 Mhz clock frequency, 8 GB PCM capacity with 2 banks in a rank, all timing parameters are based on LPDDR2 (tRCD: 50 ns, tWP: 1us) as defined in NVMain, memory controller with 64-entry buffer and FR-FCFS scheduler		
DRAM Cache	read latency: 10 ns, write latency: 10 ns, write policy: write-back, replacement policy: least recently used (LRU)		

TABLE 2
Information of Synthesized Workloads

Name	Included benchmarks
mix1	*lbm, leslie3d, astar, gcc*
mix2	*lbm, leslie3d, astar, bzip2*
mix3	*leslie3d, astar, bzip2, gcc*
mix4	*astar, bzip2, gcc, GemsFDTD*
mix5	*mcf, lbm, gcc, bzip2*
mix6	*mcf, gcc, GemsFDTD, povray*
pmix1	*B-tree, hash-map, queue, skip-list*
pmix2	*queue, B-tree, RB-tree, skip-list*
pmix3	*hash-map, queue, RB-tree, skip-list*
pmix4	*B-tree, hash-map, RB-tree, skip-list*

TABLE 3
Implemented Persistent Data Structures

Name	Description of behavior
Queue	Enqueue and de-queue nodes among 128 queues randomly
Hash-map	Insert and delete hashed keys among 128 hash tables randomly
B-tree	Insert and delete tree nodes among 128 trees randomly
RB-tree	Insert and delete tree nodes among 128 trees randomly
Skiplist	Insert and delete list nodes among 128 lists randomly

trace file. Therefore, an interconnect module containing a 32-entry buffer is used to synchronize the behavioral differences between the two systems as illustrated in Fig. 2.

2.4 Workloads

As shown in Table 2, six out of the ten workloads get mixed, from the eight benchmarks of SPEC CPU 2006, to fit in the number of simulated cores. The benchmarks are chosen according to their MPKIs (cache misses per thousand instructions) to have a large bandwidth from CPU to PCM to simulate the case of serving multiple clients in a server as in [41], [42], [43]. In this study, applications with high MPKIs *lbm*, *leslie3d*, *mcf*, *gcc* (with *s04* option), and *GemsFDTD* comprise highly stressful workloads for a PCM. The remaining benchmarks are selected for simulating the general-purpose system that runs both high and low MPKI applications simultaneously. Because there are four cores in the simulated system, as mentioned in Table 1, four benchmarks are mixed to form a single workload that is executed in a parallel manner. Consequently, both stressful and mild workloads can cover a wide range of applications.

As shown in Table 3, persistent data structures, B-Tree, RB-Tree, queue, Hash-map, and skip-list programmed with cache line flush and memory fence operations, are further implemented, where each of them contains 128 data structures and performs random insertion and deletion [5]. These data structures are similar to those in the previous papers for persistent operations [5], [44], [45], and skip-list is a basic data structure of *ZSET* (or *sorted sets*)) in Redis [46]. Finally, the data structures are combined to make four persistent workloads (prefixed as pmix) for evaluation to simulate realistic workloads in servers as shown in the shaded rows of Table 2.

2.5 Read-Modify-Write

In the darkly shaded region of Fig. 2, the RMW behaves as a front-end module and processes the command from the CPU. It first reads multiple data blocks as a transaction unit (or a row buffer size) of the PCM for each access, where each data block has a length of 64B. If the type of the input command is read, it directly responds to the desired block with the block offset indicated by the transaction address. On the other hand, the write command requires to overwrite the desired block in the prior read data first, and then writes the whole row buffer sized data back to the demanding physical address. From the output of the RMW, the controller converts the command into atomic commands for device access, such as pre-charge, activation, read, and write.

Besides, the maintenance of consistency in the case of power-off failure or a system crash is not considered further as it is beyond the scope of this paper [5], [9], [13], [45], [47]. This paper assumes that the control system of the PCM and the queues in RMW are supported by a super-capacitor as in previous works. This allows the in-flight data can be flushed in the case of a system crash [5], [45].

2.6 Motivation

Clearly, the RMW module in the PCM-based system results in a performance degradation and substantially lowers the reliability due to read disturbance introduced by redundant read operations [30]. Furthermore, many experiments held in previous PCM-related studies assume that the cache line size, commonly 64B for a general-purpose processor, matches the row buffer size of the PCM device without considering the non-symmetric case for practical usage. Thus, it is crucial to taking both factors into account to show a practical PCM-based system equipped with RMW. To resolve the above-mentioned problems, an architecture incorporating the private DRAM cache of the PCM as a part of the RMW is proposed and to further boost-up the performance with a simple operation to minimize the read-modify-write operations.

3 Baseline of RMW

The baseline model of the RMW is depicted in Fig. 3a with details. Fig. 3b shows 3 bits of flags used in this paper, *READ, WRITE,* and *WAS WRITE*. For *READ* and *WRITE*, these are used to indicate types of commands which are carried on command lines to signal read-enable and write-enable in DDR interface, respectively. For *WAS WRITE*, it means that the original command is *WRITE* after the type conversion (see step-1 in the next paragraph). When a command from the LLC is delivered to the RMW, the following steps are conducted:

1) The command is first delivered to the read request generator, which converts the types of all the incoming requests to *READ*. According to the original type of the command:
   - If the type is *READ*, a newly defined flag, *WAS WRITE*, is set to 0 (See 64-bit defined flags in Fig. 3b which can be overloaded on a memory transaction).
   - If the type is *WRITE*, *WAS WRITE* is set to 1 to remember its original command as *WRITE*.

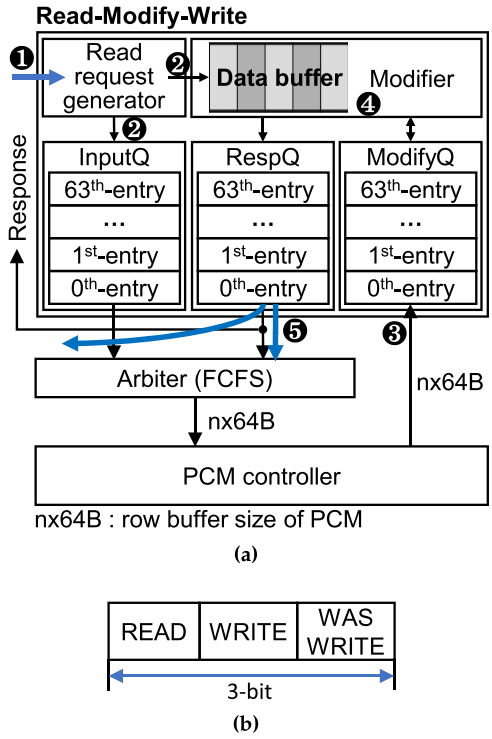

Fig. 3. (a) Baseline RMW module, and (b) flags bits of memory command in this paper.

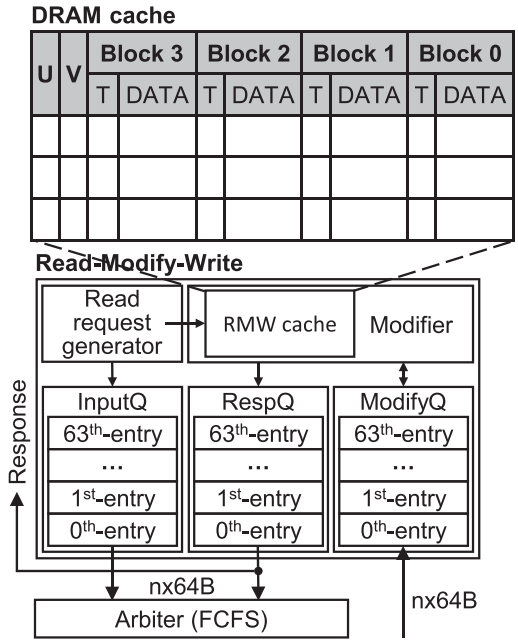

Fig. 4. Structure of the proposed cache and its position in the RMW module.

The command conversion is achieved with a concise bit flipping logic, spending no more than one clock cycle in the baseline system.

2) The converted command is stacked to the input queue (denoted by *InputQ*) when waiting for the dispatch, whereas the command data is stored in the data buffer if it is *WRITE*.
3) When the dispatched command in the previous step comes back with read data as the size of the row buffer size in the PCM, it is stacked to the modification queue (denoted by *ModifyQ*).
4) Flag *WAS WRITE* is then checked for proceeding to the next step according to the original command type:
   - If the bit is 1, the modification data in the data buffer overwrites the read data brought from the PCM. Subsequently, the flag bits of *WRITE* and *WAS WRITE* are set to "1" and "0", respectively.
   - If the bit is 0, one of the data blocks is selected according to the transaction address because the original type of the command is *READ*.

The command is then passed to the response queue (denoted by *RespQ*).

5) The command in the *RespQ* prepares write-back or read-response depending on the original command type. As both the write-back command and the request from the *InputQ* access the same input port of the memory controller, they are arbitrated under the first-come-first-serve (FCFS) policy.

4 Utilization of DRAM Cache for RMW

Motivated from Section 2, an RMW architecture that fully interacts with the DRAM cache in a PCM-based memory system is proposed. The proposed work is built upon the baseline RMW described in the previous section. The proposed architecture is called *cache-based RMW* hereafter in this paper.

4.1 Architectural Design

Fig. 4 shows an example organization of the proposed cache-based RMW. A 256B row buffer is used in this design so that each cache entry consists of four data blocks (i.e., cache lines from LLC). T-bit implies the type of a command accessing one of the data blocks, and *DATA* is a temporal field storing valid data. Therefore, the proposed cache does not need the data buffer in Fig. 3. Besides, each entry additionally requires two flag bits, V-bit and U-bit, for managing the buffered data:

- V-bit: It shows the data validity of an entry, which can be indicated with one bit because the whole row buffer data is fetched together by request.
- U-bit: It means that the entry is under update on the PCM. It prevents writing or reading to/from the entry with the addresses having different block offsets.

The repeated field structure in the cache shown in Fig. 4 makes it possible for the RMW to pre-fetch and store neighboring data so that a single command is enough to acquire the demanding data block when required. Moreover, the read reliability of the system can be enhanced when the RMW interacts with the DRAM cache.

Because the cache is implemented with a DRAM, an address decoder is needed to decode the command address to the index of the cache. The decoder is built with a lookup table (LUT), as illustrated in Fig. 5. The LUT receives the address (except for block offset) as the input and generates the index to the cache as the output. Thus, the LUT has the same number of entries as the cache does. The demanding cache index is determined by comparing the command address with the *Tags* in the LUT with a set of XOR gates. Subsequently, the concatenated output of all the XOR gates

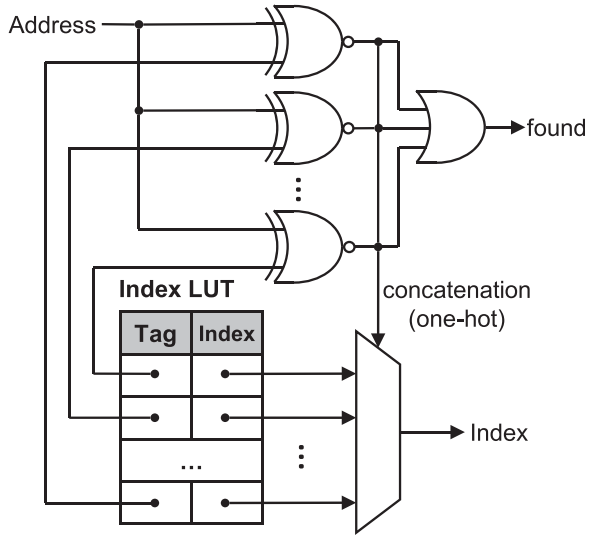

Fig. 5. Address decoder of the cache in RMW.

Algorithm 1 Process in RMW cache

```
Input: For the command on the head of InputQ
found = AddressDecoder(command.address)
if found
   if !found_entry.V
      wait for the response of previous command
   else if command.type == WRITE
      if !found_entry.U
         found_entry.dirty ← TRUE
      else
         wait for the response of command
   else
      put the command in RespQ
else
   add an entry to the cache with LRU policy
   new_entry.U ← TRUE
   new_entry.V ← FALSE
   Issue command in the next cycle
```

Fig. 6. Pseudo code of the proposed cache-based RMW.

becomes a one-hot code for index multiplexer, which is continuously fed to the address port of the cache. Besides, the existence of the requested data in the cache is confirmed by summing up the XOR results as the *found*-flag.

The concept of the RMW described in this section is somewhat similar to the RMW operations for DRAM access although there exists a slight difference for handling a PCM instead of a DRAM. For example, the difference from the RMW in [27] is as follows:

- RMW in [27]: the system defines DRAM as a main memory, so it additionally needs a data buffer for temporarily storing the write-data and flush the data once the command is processed.
- RMW in the proposed method: it places a PCM as a main memory and additionally uses a private DRAM cache for the PCM. It leverages the cache as a table for the RMW including the data buffer and reuses the data instead of flushing it right away, by which the area for the data buffer is saved with the existing resource.

4.2 Algorithm

A pseudo code is presented to show the command process of the proposed structure as described in Fig. 6. The command process is performed in two different manners according to the *found*-flag:

1) If *found* is "0" (miss), the command is inserted to an available entry in the cache according to the LRU policy as shown in Table 1. The [V, U]-bit pair is set to "01". If the replacement does not occur immediately, the command stays in the *InputQ* and waits for the availability of the cache entry.
2) If *found* is "1" (hit), the command is processed in one of the three possible manners according to its type and status:
   - If the entry is valid (V-bit=1) and the command is *READ*, the command is directly responded to the host CPU where the status of U-bit is reset to "0".
   - If the entry is valid and the command is *WRITE*, the *dirty bit* is set to 1 for writing data back to the PCM when newly written data replaces the data. The U-bit is asserted to avoid a write-after-write (WAW) data hazard.
   - If the entry is invalid (V-bit=0), it means that the entry is under update (U-bit=1). Thus, the command stays in the *InputQ* and waits for the response of the previous command.

When the first read command returns from the PCM, the data field of the corresponding entry is loaded with the read data and the valid bit (V-bit) is set to "1". The U-bit is reset to "0" because the entry update is complete.

Since the proposed design frequently offloads redundant read operations, the read disturbance is also reduced significantly. Depending on the characteristics of workload, it may not offer a noticeable improvement in the operation speed. For example if a command with cache miss is under process on the PCM device, all the commands behind the miss-command in the *InputQ* are constrained in the queue. This problem is critical for large miss penalty (called a stuck-in-queue problem). In particular, if the locality of an application is low, frequent cache misses results in a performance degradation. A novel operation to effectively mitigate the problem is explained in the next section.

5 Typeless Command Merging

5.1 Architectural Design

To mitigate the stuck-in-queue problem, the *typeless merge* operation, which merges commands without any regard to the command type, is proposed to drain commands clogged in the *InputQ*. Fig. 7 shows the details of the new entry structure and the modified RMW along with an example, which will be discussed in the sub-section below. As shown in the figure, M-bit is an additional bit in each block field of the cache. It indicates an entry that represents multiple commands accessing the same address but different block offsets (i.e., different cache lines). For example, if the M-bits in block 0 and block 2 are set to 1, they would represent two commands generated by a CPU and will be merged into a single command regardless of it being *READ* or *WRITE*. To make this new operation executable, this paper incorporates a *Merger* and a *De-merger* for merging input commands from

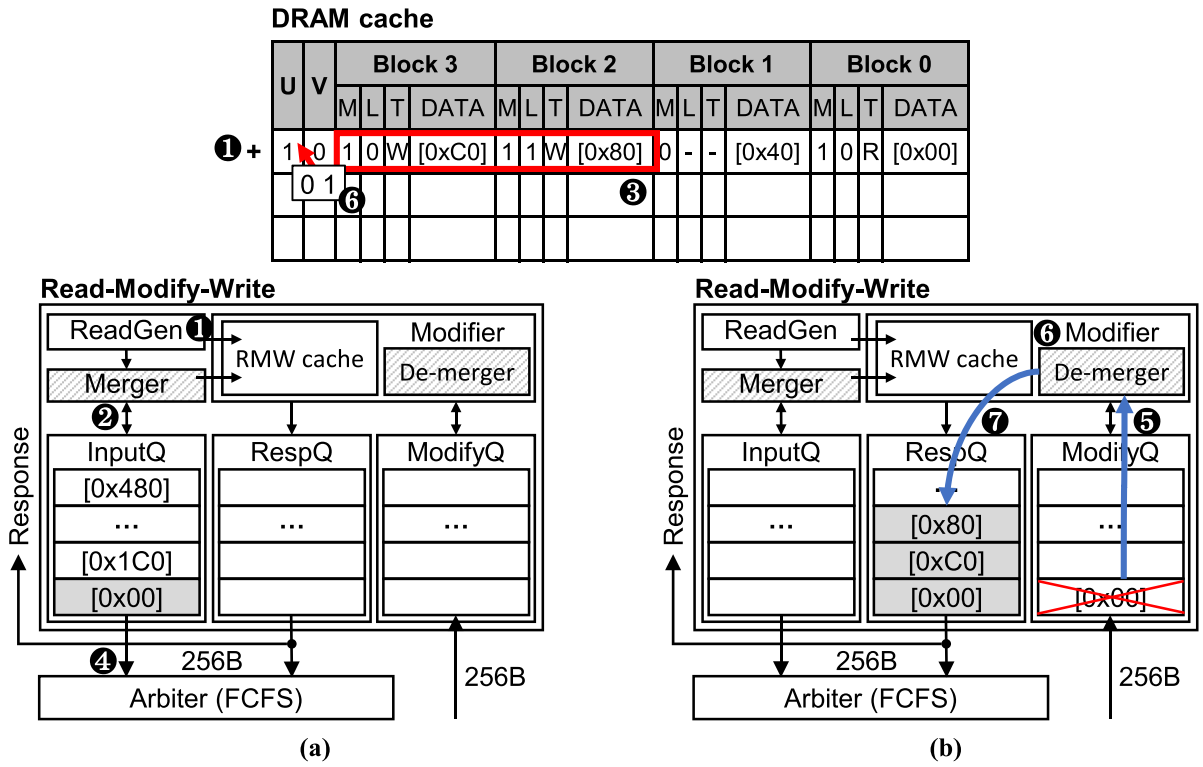

Fig. 7. Cache table and RMW module supporting merge operation. (a) Merge process, and (b) De-merge process.

the processor and disassembling the merged command responded from the PCM, respectively.

5.2 Algorithm

For the merge operation, Algorithm 1 in Fig. 6 is slightly modified as shown in Fig. 8. Additional pseudo codes supporting the merge operation are illustrated in Fig. 9.

Modification of Algorithm 1. The operation handling the false case of the V-bit in Algorithm 1 is slightly modified (see Fig. 8). The command on the head is merged with the accessing entry if it is invalid (V-bit=0) disregarding the state of the U-bit. This is because the merged command is to be split into original commands again and the corresponding responses are to be responded to the CPU. Since there is only one "if" case added to the algorithm, it is implemented using a 2-input multiplexer with a slight modification in the hardware.

Algorithm 1 Process in RMW cache (modified)

```
Input: For the command on the head of InputQ
found = AddressDecoder(command.address)
idx = ExtractBlockOffset(command.address)
if found
   if !found_entry.V
      if !found_entry[idx].M
         found_entry[idx].M ← TRUE
         Record data if command.type == WRITE
      else
         wait for the response of prev. command
   else if command.type == WRITE
      if !found_entry.U
         found_entry.dirty ← TRUE
      else
         wait for the response of command
   else
      put the command in RespQ
else
   add an entry to the cache with LRU policy
   new_entry.U ← TRUE
   new_entry.V ← FALSE
```

Fig. 8. Modified version of Algorithm 1 for merge operation in which the modified part is shaded.

Algorithm 2 *Merger*

```
Input: InputQ
head = InputQ.head
if pending cycle ≥ pending threshold
   dispatch the request right away
for each cmd in InputQ do
   if same address && different block offset
      idx = Extract-Block-Offset(cmd.address)
      if !latched_entry[idx].M
         latched_entry[idx].M ← TRUE
         Record data if command.type == WRITE
end for
```

Algorithm 3 *De-merger*

```
Input: ModifyQ, latched cache entry
head = ModifyQ.head
for each block in latched_entry do
   if block.M
      block.M ← FALSE
      Generate a response satisfying block info
      push.the response into RespQ
end for
latched_entry.U ← FALSE
latched_entry.V ← TRUE
```

Fig. 9. Pseudo code of *Merger* and *De-Merger*.

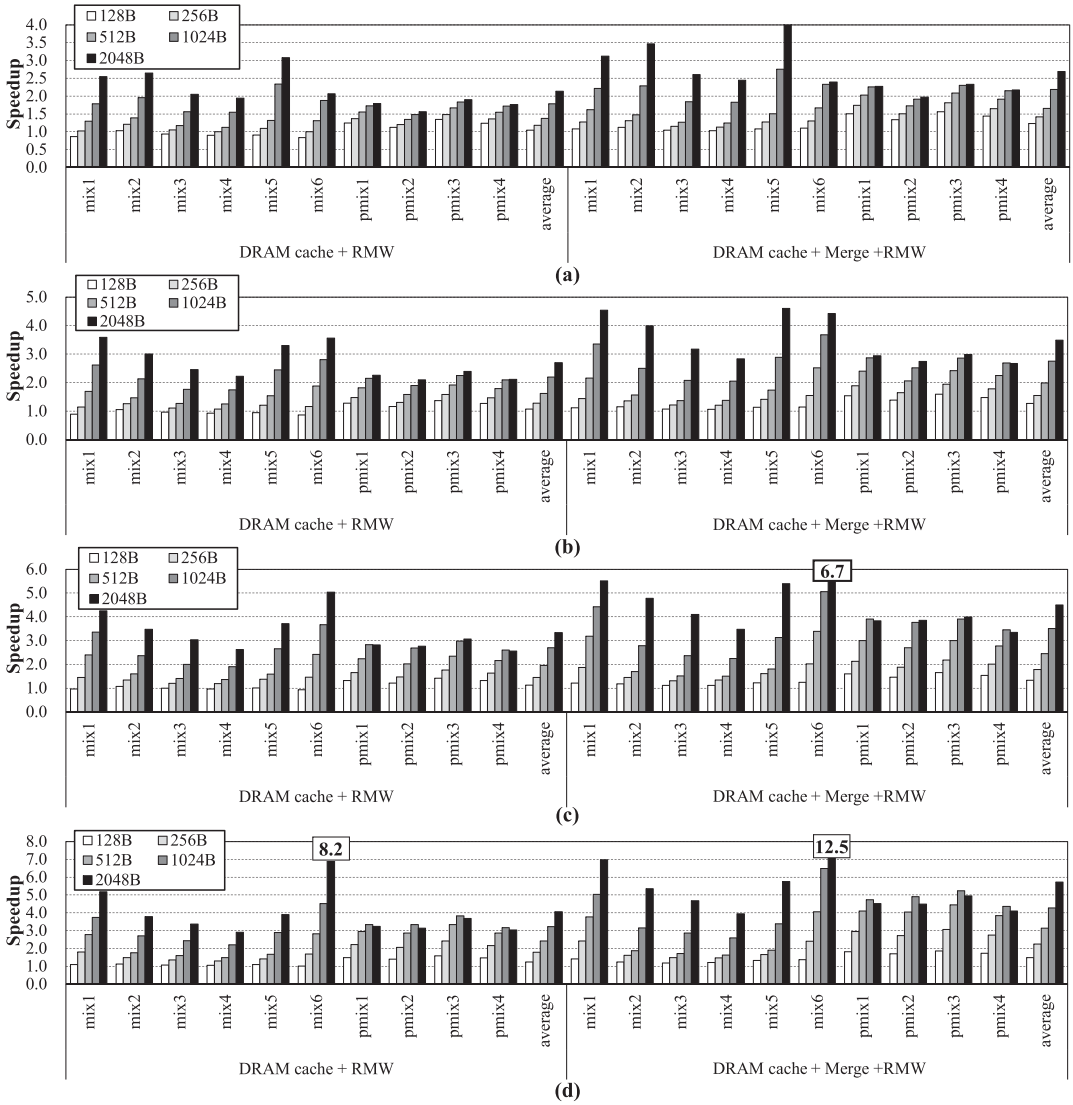

Fig. 10. Speedup comparisons between the RMW only applying DRAM cache and the RMW applying typeless merge operation with respect to different DRAM cache entry numbers. Some bars out of range of the figure are labelled with specific values. (a) 512-entry, (b) 1024-entry, (c) 2048-entry, (d) 4096-entry.

Merger (Algorithm 2). As shown in Fig. 9, the *Merger* ensures that the commands generating new entries in the cache are not issued immediately. It waits for commands that access the same address but different block offsets. As shown in the algorithm, the waiting time is determined by the pending threshold which is chosen by the experiments. If there is no command for merging within the pending threshold in *InputQ*, it is dispatched right away for PCM access. When searching for the commands for merging, the *Merger* identifies the commands satisfying the conditions in the pseudo code and sets all the M-bits to "1", which means that the command is merged with that entry. Because the commands are merged into one entry, the throughput of the system can be significantly improved. As a result, the *Merger* can hide the miss penalty at the expense of simple logics as shown in the algorithm.

De-merger (Algorithm 3). The merged commands that are returned from the PCM must be retrieved with the information recorded in the cache. As demonstrated in Fig. 9, the *De-merger* checks the M-bit of each block field in the entry and retrieves the commands merged in the *InputQ* with the assertion of the V-bit. Finally, all the disassembled commands are pushed into the *RespQ* for responses. Moreover, the ordering can also be maintained if an additional L-bit is added to each block field of the entry. The description for this bit is given in the following sub-section.

5.3 Example

In this example, a PCM with a row buffer size of 256B is assumed, and the command at the head of the *InputQ* is accessing address 0x0. The pending threshold of the *Merger* is chosen as eight. For better understanding, Fig. 7a illustrates the work-flow of merging:

1) 0x0-command first generates a new entry for the cache due to cache miss. During the generation, the M-bit is set to 1 to indicate that the block is occupied.
2) Subsequently, commands 0xC0 and 0x80 are delivered into the *InputQ* within the pending cycle of the *Merger*. Concurrently, the *Merger* matches the address of the incoming command and the cache entries. The L-bit of 0x80-command is set to 1 to indicate that it is the latest *WRITE* command.

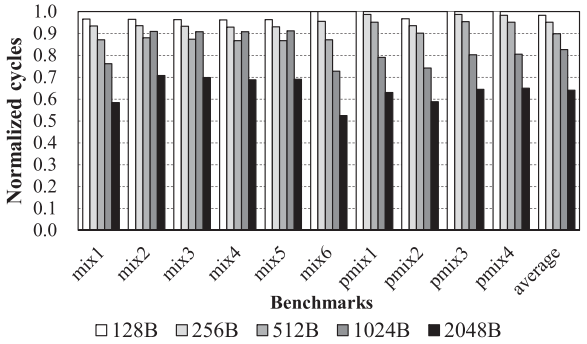

Fig. 11. Speed degradation rate with respect to different row buffer size where 64B is the baseline. The lower rate the worse performance.

3) The data of the commands are recorded into the generated entry in step-1 with assertions of the M-bit and the U-bit.
4) After the pending cycle exceeds, the 0x0-command is dispatched to the memory controller.

When the response of the 0x0-command is returned to the RMW, it is first queued into the *ModifyQ* for the de-merging process. Fig. 7b depicts the process of de-merging:

5) The 0x0-command at the head of the *ModifyQ* is directed to the *De-Merger* for decomposition.
6) All the M-bits in the cache are identified to determine the merged commands. Although two *WRITE* commands are merged in the entry, it is unnecessary to issue all of them to the PCM because this may be inconsistent with the purpose of the merge operation. Thus, the last *WRITE* command merged is issued, which is implied by the L-bit. Meanwhile, the [U, V]-bit pair is set to "01" after the decomposition.
7) Finally, the decomposed commands are stored into the *RespQ* for direct response or write-back.

This example shows that the merge operation "virtually" dispatches three commands in the eight pending cycles with a single representative command. Therefore, the throughput of the memory system is significantly improved with a simple operation while maintaining the read reliability due to the reduction of redundant read operations.

6 Evaluation

6.1 Speedup

The speed of the baseline RMW is degraded as the row buffer size increases because a memory transaction of a larger buffer size requires a longer burst length. The simulation results shown in Fig. 11 verify the degradation. The proposed cache-based RMW, however, achieves a speedup with an increase in the row buffer size. In each graph in Figs. 10a, 10b, 10c, and 10d, the left sub-graph shows the result with the cache-based RMW whereas the right sub-graph represents the improvements made by the merging operations. The baseline for normalization is the model having a 64B-row buffer size. As shown in the left sub-figures of Figs. 10a, 10b, 10c, and 10d, the speedup increases as the cache capacity enlarges thanks to the enhanced cache hit rate. The average speedups of 512B-row buffer systems are 1.4 times, 1.6 times, 1.8 times, and 2.4 times when there are 512, 1,024, 2,048, and 4,096 entries in the DRAM cache, respectively. The maximum improvement is 4.1 times compared to the baseline on average, when the entry number is 4,096 and row buffer size is 2 KB.

Some benchmarks have speedups smaller than 1 when the entry number is smaller than 1,024 for 128B row buffer. This degradation occurs due to the "stuck-in-queue" problem as discussed in Section 4.2. To mitigate stuck-in-queue problem, the merge operation mentioned in Section 5 is applied to the cache-based RMW. The right sub-figures of Figs. 10a, 10b, 10c, and 10d show the speedup of merge operation ranging from 128B to 2KB row buffer size. As shown in these figures, the merge operation enhances the performance markedly. For a system with 512B-row buffer, the speedup achieves 1.7, 2.0, 2.5, and 3.2 times when the numbers of the cache entries are 512, 1,024, 2,048, and 4,096, respectively (see Figs. 10a, 10b, 10c, 10d). The maximum speedup is 5.7 times on average with 2KB-row buffer and 4,096-entry cache, which is higher than the value of 4.1 times when using DRAM-only method.

6.2 Benefits of Reliability

6.2.1 Read Disturbance

The proposed system provides performance improvement, and yet it is still insufficient to indicate that the proposed method is suitable for general use concerning reliability. Another unexpected benefit of the cache-based RMW is device reliability. Read disturbance is an error that is incurred by frequent and redundant read operations on a specific cell. Since a cache filters out most of the commands accessing neighboring addresses to prevent the commands from entering the PCM device, the read disturbance can be highly reduced by applying the proposed method.

To cover technology variations, six different bit error rates are chosen as 1E-2, 1E-3, 1E-4, 1E-5, 1E-6, and 1E-7. Fig. 12 shows the average normalized bit errors which represents the ratio of the bit errors generated by proposed RMW against the baseline. These values are represented by the bar graphs. On the other hand, the line graphs in the figure show the average numbers of errors occurred per read operation with respect to the size of row buffer. The horizontal axis represents the row buffer size whereas four cache sizes are chosen as 512, 1,024, 2,048, and 4,096 entries, respectively. The four graphs in Fig. 12 represent the results for these four cache sizes, respectively. In this figure, although the errors per read operation increase due to the fast decrease of the total read operations (or dominator), the number of bit errors decrease with the increase of the row buffer size, from which the proposed method consequently enhances the overall read reliability compared to the baseline. This is because the proposed model allows the controller to pre-fetch and reuse the data as the size of the row buffer increases. For 512B row buffer tested at the bit error rate of 1E-5, for example, the average errors are reduced by 41, 39, 40, and 49 percent when the cache has 512, 1,024, 2,048, and 4,096 entries, respectively, as compared to that in the baseline.

6.2.2 Cell Endurance

The lifetime is another important measurement of PCM characteristic which is estimated by measuring the number of writes to the hottest position (or the worst-case wear counts), by which it is used for quantifying the effectiveness of wear-leveling in [48]. Fig. 13a shows the writes to the

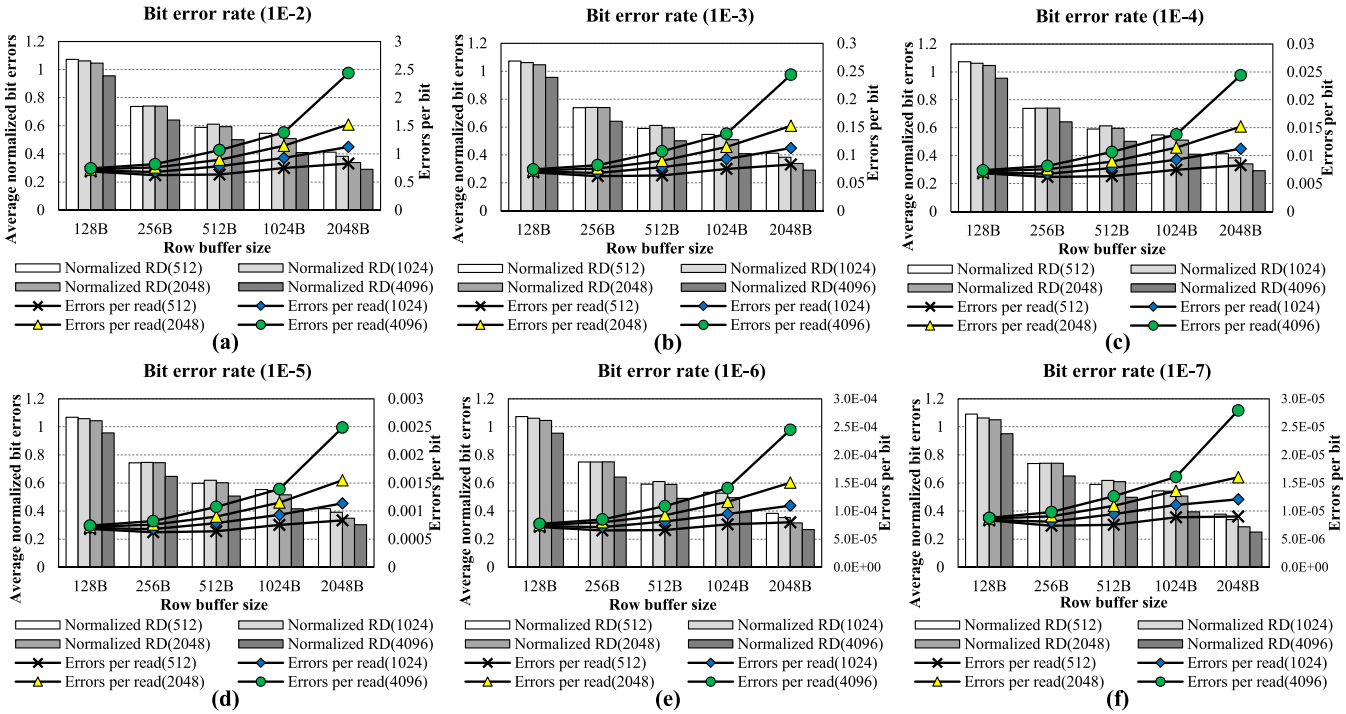

Fig. 12. Average normalized bit errors and average bit errors/read command after applying merge operation with different DRAM cache entries (512-4096 entries). (a) BER=1E-2, (b) BER=1E-3, (c) BER=1E-4, (d) BER=1E-5, (e) BER=1E-6, (f) BER=1E-7.

hottest position of the proposed method equipped with a 4,096-entry DRAM cache which is normalized to the baseline by increasing the row buffer size from 128B to 2 KB. The wear-out becomes worse if the row buffer is enlarged from 128B to 512B; however, it can be handled as similar to the baseline by modulating the row buffer size larger than 1 KB due to data pre-fetching effect on the DRAM cache. The proposed method with 2 KB row buffer and 4,096-entry DRAM achieves to 39 percent reduction of the worst-case wear counts as compared to the baseline. Fig. 13b also shows that the proposed method prevents wear-out more effectively as the DRAM cache size increases. For a 512B row buffer, the wear-outs to the hottest cell are reduced by 24.3 percent as the entry number increases from 512 to 4,096.

6.3 Energy Consumption and the Selection of the Optimal Row Buffer Size

Fig. 14 shows that the average energy consumption of a memory system generally increases with the row buffer size by adopting the proposed merge-operation with respect to a different DRAM cache size, where the energy consumptions of different row buffer sizes are measured for each DRAM cache size. The horizontal red line shows the energy consumption of the baseline. The energy consumption gradually decreases as the number of DRAM cache entries increases. For a 4,096-entry DRAM cache 512B row buffer,

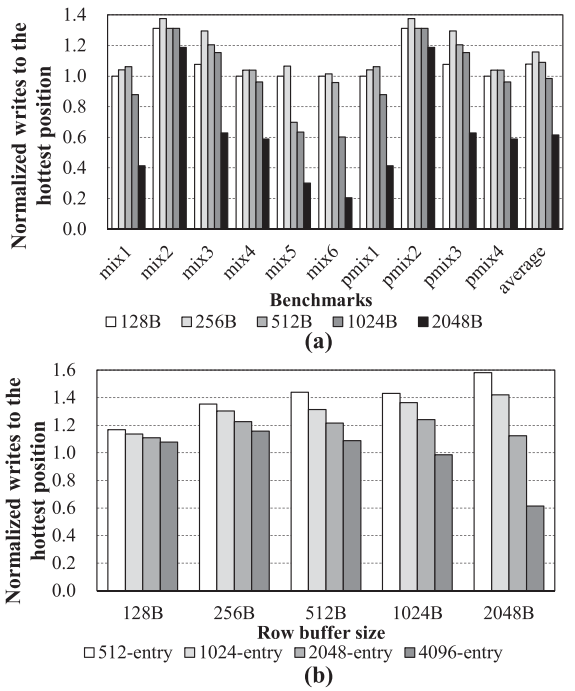

Fig. 13. Normalized writes to the hottest position of the proposed merge-operation: (a) the results for each benchmark when DRAM cache has 4096 entries, (b) the results for different DRAM cache entries.

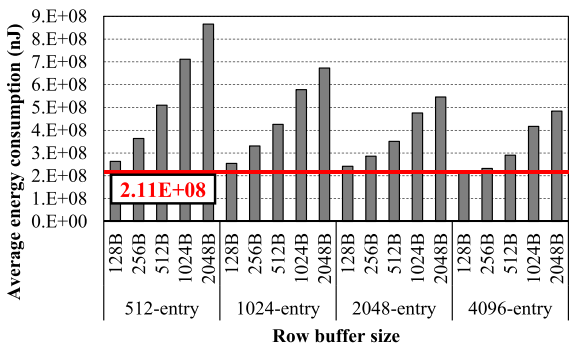

Fig. 14. Average energy consumption of the proposed merge-operation where the red line shows the average energy consumption of the baseline in this paper.

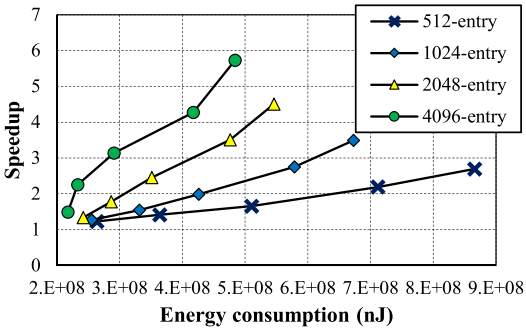

Fig. 15. Relationship between energy consumption and speedup by modulating the row buffer size for different cache entry numbers.

the increase of the energy consumption by the proposed merge-operation is about 38 percent while the speed is improved by 3.2 times compared to the baseline. This increase of energy consumption is relatively small because the main memory consumes about 11 percent of a whole computing system [49]. As a result, the proposed method can achieve $\times 3.2$ of the speedup with a relatively small increase in energy consumption (about $4\% = 38\% \times 11\%$) compared to the baseline.

The optimal row buffer size should be chosen to achieve the best trade-off between the energy consumption and speedup. Fig. 15 shows the relationship between the energy consumption and speedup by modulating the row buffer size from 128B, 256B, 512B, 1,024B, to 2,048B. The five symbols in each graph represent these sizes. Since the energy consumption increases by nearly 1.5 times when row buffer size is above 1,024B as compared to the baseline which too large to be adopted in the system, 512B is chosen as the optimal row buffer size. This selection is compliant with the previous research that also chooses the transaction unit to be no more than 512B [2], [21], [22]. The selected trade-off increases the speed by 3.2 times and reduces total error occurrences by 49 percent, as it increases the energy consumption by 38 percent when compared to the baseline for 4,096-entry DRAM cache with merge operation. The hardware cost of the merge operation very small because 4,096-entry DRAM cache demands about 2 MB while a DRAM requires approximately 5.5$/GB [9].

6.4 Comparison with Related Works

For fairly comparison among different performance enhancement schemes, this paper applies [12], [32], and [34] to the RMW integrated with the DRAM cache instead of nave RMW. The average speedups and average energy consumptions of all schemes including the proposed method (*typless* merge operation) are shown in Figs. 16a and 16b, respectively. The graphs denoted by NonBlockBank, WSHR, and FgNVM, represents the schemes of [12], [32], and [34], respectively. For speedup, the proposed method achieves the best performance as it offers 19.2 percent improvement on average as compared to the others when the row buffer size is 512B (the optimal row buffer size discussed in the previous sub-section). For the energy consumption in Fig. 16b, all schemes show similar energy consumptions, whereas the proposed method consumes 1 percent smaller consumption on average as compared to the others due to the decrease of command processing command processing.

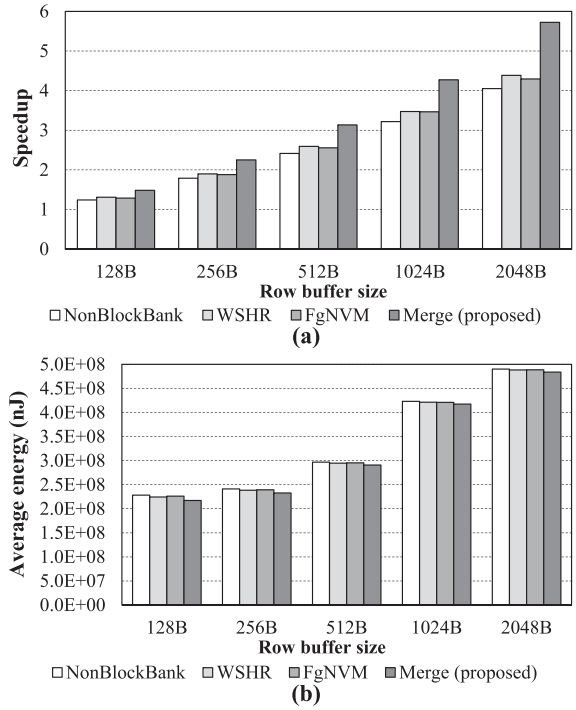

Fig. 16. Comparison with different performance enhancement schemes where all results are normalized to the baseline in the paper. (a) speedup, (b) energy consumptions.

7 Conclusions and Future Work

A PCM is an indispensable memory device as data access becomes increasingly intensive with an increasing demand for data reliability. It is necessary to increase the size of the row buffer in a PCM device to larger than 128B to address the drawbacks of the PCM compared to DRAM. This paper first proposes an RMW architecture utilizing a DRAM cache for enhancing read reliability and the performance in a PCM-based system. However, it may induce an unexpected performance degradation for some workloads due to high miss penalty of the cache in the PCM. Therefore, *merge*, an operation for combining the cache and RMW, is proposed, not only to compensate for the throughput reduction as much as possible but also to achieve even improved performance. As a result, the merge operation adopted in the 512B-row buffer system with 4,096-entry DRAM cache enhances the speed and read reliability by 3.2 times and 49 percent, respectively, on average. The DRAM cache requires approximately 2 MB of storage space, of which cost is very small as discussed in the previous section.

This work assumes that the memory controller of the PCM including the RMW is supported by a super-capacitor so that the proposed model offloads the work of persistent model as much as possible. In the future, to further reduce the energy consumption and area utilization consumed by the heavy super-capacitor, we need to devise a much more concrete model which can deal with the persistent model without incorporating the super-capacitor.

Acknowledgments

This paper was result of the research project supported by SK Hynix Inc. and was supported by the Technology Innovation Program (10080613, DRAM/PRAM heterogeneous

memory architecture and controller IC design technology research and development) funded By the Ministry of Trade, Industry & Energy (MOTIE, Korea).

REFERENCES

[1] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 256–267.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 2–13.

[3] S. Mittal, "A survey of soft-error mitigation techniques for non-volatile memories," *Comput.*, vol. 6, no. 1, 2017, Art. no. 8.

[4] Intel, "Intel and micron produce breakthrough memory technology," 2015. [Online]. Available: https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology

[5] S. Shin, J. Tuck, and Y. Solihin, "Proteus: A flexible and fast software supported hardware logging approach for NVM," in *Proc. 50th Annu. Int. Symp. Microarchitecture*, 2017, pp. 178–190.

[6] S. Sundararaman, N. Talagala, D. Das, A. Mudrankit, and D.Arteaga, "Towards software defined persistent memory: Rethinking software support for heterogeneous memory architectures," in *Proc. 3rd Workshop Interactions NVM/FLASH Operating Syst. Workloads*, 2015, pp. 6:1–6:10.

[7] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proc. 9th Eur. Conf. Comput. Syst.*, 2014, pp. 15:1–15:15.

[8] B. Bhattacharjee, M. Canim, C. A. Lang, G. A. Mihaila, and K. A. Ross, "Storage class memory aware data management," *Bulletin IEEE Comput. Soc. Tech. Committee Data Eng.*, vol. 33, no. 4, pp. 35–40, 2010.

[9] J. Huang, K. Schwan, and M. K. Qureshi, "NVRAM-aware logging in transaction systems," *Proc. VLDB Endowment*, vol. 8, pp. 389–400, 2014.

[10] J. Guerra, L. Mrmol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei, "Software persistent memory," in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 22–29.

[11] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proc. 39th Int. Symp. Comput. Archit.*, 2012, pp. 380–391.

[12] P. Zhou, J. Y. B. Zhao, and Y. Zhang, "Throughput enhancement for phase change memories," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 2080–2093, Aug. 2014.

[13] F. Xia, D. Jiang, J. Xiong, and N. Sun, "HiKV: A hybrid index key-value store for DRAM-NVM memory systems," in *Proc. USENIX Annu. Techn. Conf.*, 2017, pp. 349–362.

[14] Numonyx, "Numonyx," 2009. [Online]. Available: http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf

[15] AMD, "High-bandwidth memory reinventing memory technology," 2015. [Online]. Available: https://www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf

[16] R. Fisher, "Optimizing NAND flash performance," in *Flash Memory Summit*, Santa Clara, U.S., Aug. 2008. [Online]. Available: https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2008/20080812_F2B_Fisher.pdf

[17] C. Kim, J. Ryu, T. Lee, H. Kim, J. Lim, J. Jeong, S. Seo, H. Jeon, B. Kim, I. Lee, D. Lee, P. Kwak, S. Cho, Y. Yim, C. Cho, W. Jeong, J. Han, D. Song, K. Kyung, Y. Lim, and Y. Jun, "A 21nm high performance 64Gb MLC NAND flash memory with 400MB/s asynchronous toggle DDR interface," in *Proc. Symp. VLSI Circuits-Digest Tech. Papers*, 2011, pp. 196–197.

[18] J. Kim, S. Kim, and J. Kim, "Subpage programming for extending the lifetime of NAND flash memory," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2015, pp. 555–560.

[19] M. Kim, J. Lee, S. Lee, J. Park, and J. Kim, "Improving performance and lifetime of large-page NAND storages using erase-free subpage programming," in *Proc. 54th Annu. Des. Autom. Conf.*, 2017, pp. 1–6.

[20] N. H. Seong, D. H. Woo, and H. H. Lee, "Security refresh: Protecting phase-change memory against malicious wear out," *IEEE Micro*, vol. 31, no. 1, pp. 119–127, Jan./Feb. 2011.

[21] 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 *Proc. 42nd Annu. Int. Symp. Microarchitecture*, 2009, pp. 14–23.

[22] F. Huang, D. Feng, W. Xia, W. Zhou, Y. Zhang, M. Fu, C. Jiang, and Y. Zhou, "Security RBSG: Protecting phase change memory with security-level adjustable dynamic mapping," in *Proc. 30th IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 1081–1090.

[23] H. Yu and Y. Du, "Increasing endurance and security of phase-change memory with multi-way wear-leveling," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1157–1168, May 2014.

[24] J. Yun, S. Lee, and S. Yoo, "Dynamic wear leveling for phase-change memories with endurance variations," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 9, pp. 1604–1615, Sep. 2015.

[25] Intel Corporation, "ECC Handling Issues on Intel XScale I/O Processors", Tech. Note, 1st ed., Dec. 2003. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/application-notes/ecc-handling-issues-on-xscale-io-processors-note.pdf

[26] J. H. Edmondson, R. A. Alfieri, M. F. Harris, and S. E. Molnar, "Coalescing to avoid read-modify-write during compressed data operations," US Patent 8928681 B1, Jan. 2015.

[27] P. R. Hiller, W. P. Hovis, and J. A. Kirscht, "Memory controller and method for optimized read/modify/write performance," US Patent 7908443 B2, Mar. 2011.

[28] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proc. 18th Int. Symp. High Perform. Comput. Archit.*, 2012, pp. 201–210.

[29] 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 *Proc. 16th Int. Symp. High-Perform. Comput. Archit.*, 2010, pp. 1–11.

[30] N. Castellani, G. Navarro, V. Sousa, P. Zuliani, R. Annunziata, M. Borghi, L. Perniola, and G. Reimbold, "Comparative analysis of program/read disturb robustness for GeSbTe-based phase-change memory devices," in *Proc. IEEE 8th Int. Memory Workshop*, 2016, pp. 1–4.

[31] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 24–33.

[32] Y. Kim, S. Yoo, and S. Lee, "Write performance improvement by hiding R drift latency in phase-change RAM," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 897–906.

[33] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Moss, "Increasing PCM main memory lifetime," in *Proc. Conf. Des. Autom. Test Europe*, 2010, pp. 914–919.

[34] M. Poremba, T. Zhang, and Y. Xie, "Fine-granularity tile-level parallelism in non-volatile memory architecture with two-dimensional bank subdivision," in *Proc. 53th Annu. Des. Autom. Conf.*, 2016, pp. 1–6.

[35] M. Arjomand, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Boosting access parallelism to PCM-based main memory," in *Proc. 43th Annu. Int. Symp. Comput. Archit.*, 2016, pp. 695–706.

[36] J. Chang and A. Sainio, "NVDIMM-N cookbook: A soup-to-nuts primer on using NVDIMM-ns to improve your storage performance," 2015. [Online]. Available: http://www.snia.org/sites/default/orig/FMS2015/Chang-Sainio_NVDIMM_Cook book.pdf

[37] R. Peglar, A. Bumgarner, and T. Talpey, "Persistent memory, NVM programming model, and NVDIMMS," 2017. [Online]. Available: https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_FR21_SNIA.pdf

[38] NorthwestLogic, "AXI interface core," 2017. [Online]. Available: https://nwlogic.com/products/docs/AXI_Interface_Core.pdf

[39] NorthwestLogic, "Read-modify-write core," 2017. [Online]. Available: https://nwlogic.com/products/docs/Read-Modify-Write_Core.pdf

[40] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul.–Dec. 2015.

[41] F. Zeng, L. Qiao, M. Liu, and Z. Tang, "Memory performance characterization of SPEC CPU2006 benchmarks using TSIM," *Phys. Procedia*, vol. 33, pp. 1029–1035, 2012.

[42] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *Proc. 43rd Annu. Int. Symp. Comput. Archit.*, 2016, pp. 519–531.

[43] J. Zhao, O. Mutlu, and Y. Xie, "FIRM: Fair and high-performance memory control for persistent memory systems," in *Proc. 47th Annu. Int. Symp. Microarchitecture*, 2014, pp. 153–165.
[44] J. Coburn, A. M. Caulfield, L. M. G. A. Akel, R. J. R. K. Gupta, and S. Swanson, "NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *Proc. 16th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2011, pp. 105–118.
[45] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "ATOM: Atomic durability in non-volatile memory through hardware logging," in *Proc. 23rd Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 361–372.
[46] Redis, "An introduction to redis data types and abstractions," [Online]. Available: https://redis.io/topics/data-types-intro
[47] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," in *Proc. 41st Int. Symp. Comput. Archit.*, 2014, pp. 265–276.
[48] Y. Du, M. Zhou, B. R. Childers, D. Moss, and R. Melhem, "Bit mapping for balanced PCM cell programming," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 428–439.
[49] Intel, "Microprocessor power impacts," 2010. [Online]. Available: https://www.glsvlsi.org/archive/glsvlsi10/pant-GLSVLSI-talk.pdf
[50] C. Villa, D. Mills, G. Barkley, H. Giduturi, S. Schippers, and D. Vimercati, "A 45nm 1Gb 1.8V phase-change memory," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2010, pp. 270–271.
[51] Y. Choi, I. Song, M. H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y. J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y. T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2012, pp. 46–48.
[52] K. Wu, Y. Huang, and D. Li, "Unimem: Runtime data management on non-volatile memory-based heterogeneous main memory," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, pp. 58:1–58:14.
[53] D. Kline, R. Melhem, and A. K. Jones, "Counter advance for reliable encryption in phase change memory," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 209–212, Jul.–Dec. 2018.

Hyokeun Lee (S'19) received the BS degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2016, where he is currently working toward the integrated MS and PhD degrees in electrical and computer engineering at Seoul National University. His current research interests include non- volatile memory controller design, hardware persistent model for non-volatile memory, and computer architecture. He is a student member of the IEEE.

Moonsoo Kim received the BS degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 2014, where is currently working toward the integrated MS and PhD degrees in electrical and computer engineering at Seoul National University, Seoul, Korea. His current research interests include SoC design of video/images, and low-power, reliable design of memory hierarchy.

Hyunchul Kim received the BS degree in computer science and engineering from Dankook University, Seoul, South Korea, in 2002, where he is currently working as a senior engineer at SK Hynix, Icheon, Gyeonggi-do, South Korea.His current research interests include new memory solution architecture.

Hyun Kim (M'12) received the BS, MS and PhD degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was a BK assistant professor with the BK21 Creative Research Engineer Development for IT, Seoul National University. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is an assistant professor. His current research interests include algorithm, computer architecture, memory, and SoC design for low-complexity multimedia applications, and deep neural networks. He is a member of the IEEE.

Hyuk-Jae Lee (M'04) received the BS and MS degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1987 and 1989, respectively, and the PhD degree in electrical and computer Engineering from Purdue University, West Lafayette, IN, in 1996. From 1998 to 2001, he was a senior component design engineer with the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR. From 1996 to 1998, he was a faculty member with the Department of Computer Science, Louisiana Tech University, Ruston, LA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is a professor. He is the Founder of Mamurian Design, Inc., Seoul, a fabless SoC design house for multimedia applications. His current research interests include computer architecture and SoC for multimedia applications. He is a member of the IEEE.