Ph.D. DISSERTATION

Mitigating Disturbance Errors and Enhancing RMW Performance for PCM

상변화 메모리 시스템의 간섭 오류 완화 및 RMW 성능 향상 기법

BY

Hyokeun Lee

AUGUST 2021

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Mitigating Disturbance Errors and Enhancing RMW Performance for PCM

상변화 메모리 시스템의 간섭 오류 완화 및 RMW 성능 향상 기법

지도교수 이 혁 재
이 논문을 공학박사 학위논문으로 제출함

2021년 8월

서울대학교 대학원

전기 정보 공학부

이 효 근

이효근의 공학박사 학위 논문을 인준함

2021년 8월

위 원 장: 정 덕 균
부위원장: 이 혁 재
위 원: 김 장 우
위 원: 심 재 웅
위 원: 류 수 정

Abstract

Phase-change memory (PCM) announces the beginning of the new era of memory systems, owing to attractive characteristics. Many memory product manufacturers (e.g., Intel, SK Hynix, and Samsung) are developing related products. PCM can be applied to various circumstances; it is not simply limited to an extra-scale database. For example, PCM has a low standby power due to its non-volatility; hence, computation-intensive applications or mobile applications (i.e., long memory idle time) are suitable to run on PCM-based computing systems.

Despite these fascinating features of PCM, PCM is still far from the general commercial market due to low reliability and long latency problems. In particular, low reliability is a painful problem for PCM in past decades. As the semiconductor process technology rapidly scales down over the years, DRAM reaches 10 nm class process technology. In addition, it is reported that the write disturbance error (WDE) would be a serious issue for PCM if it scales down below 54 nm class process technology. Therefore, addressing the problem of WDEs becomes essential to make PCM competitive to DRAM. To overcome this problem, this dissertation proposes a novel approach that can restore meta-stable cells on demand by levering two-level SRAM-based tables, thereby significantly reducing the number WDEs. Furthermore, a novel randomized approach is proposed to implement a replacement policy that originally requires hundreds of read ports on SRAM.

The second problem of PCM is a long-latency compared to that of DRAM. In particular, PCM tries to enhance its throughput by adopting a larger transaction unit; however, the different unit size from the general-purpose processor cache line fur-

ther degrades the system performance due to the introduction of a read-modify-write (RMW) module. Since there has never been any research related to RMW in a PCM-based memory system, this dissertation proposes a novel architecture to enhance the overall system performance and reliability of a PCM-based memory system having an RMW module. The proposed architecture enhances data re-usability without introducing extra storage resources. Furthermore, a novel operation that merges commands regardless of command types is proposed to enhance performance notably.

Another problem is the absence of a full simulation platform for PCM. While the announced features of the PCM-related product (i.e., Intel Optane) are scarce due to confidential issues, all priceless information can be integrated to develop an architecture simulator that resembles the available product. To this end, this dissertation tries to scrape up all available features of modules in a PCM controller and implement a dedicated simulator for future research purposes.

keywords: Computer Architecture, Non-Volatile Memory, Phase-Change Memory, Writ Disturbance, Read Disturbance, Read-Modify-Write, Memory Simulator

student number: 2016-27167

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Limitation of Traditional Main Memory Systems

With the advent of in-memory database [1], [2], [3] and deep learning applications [4], the total footprint in the main memory becomes considerable in a computing system, thereby requiring more than a hundred gigabytes of main memory devices.

Although Moore's law has propelled the industry to scale down the semiconductor process technology, the law becomes no longer available due to physical rationales. For example, the DRAM of DDR5 generation still maintains a 10 nm class process technology in order to guarantee the reliability of the device (e.g., sensing margin on the cell capacitor) as DDR4 does [5].

In contrast, the byte-addressable non-volatile memory (NVM) device garners attention as the next-generation main memory because the NVM does not require capacitors that occupy a considerable area in a traditional DRAM device [6]. The cell array of the byte-addressable NVM device is comprised of resistive materials rather than the

Table 1.1: Summary of various byte-addressable memory devices

Features	**DRAM**	**STT-MRAM**	**FeRAM**	**PCM**
Mechanism	Electric charge in capacitors	Magnetized direction on magnetic materials	Polarization of ferroelectric materials	Different states with heat programming
Non-volatility	No	Yes	Yes	Yes
Cell size	$6F^2$ - $10F^2$	6 - $50F^2$	4 - $20F^2$	4 - $12F^2$
Read latency	50 ns	10 ns	10 ns	50 ns
Write latency	50 ns	50 ns	50 ns	500 ns
Endurance	>1E+15	>1E+15	1E+12	1E+8
Standby power	Refresh	No	No	No

high-latency NAND gates. Phase-change memory (PCM), spin-torque transfer magnetic RAM (STT-MRAM), and ferroelectric RAM (FeRAM) are representative candidates for the NVM-based main memory. In particular, PCM is closest to deployment in the commercial market owing to its high scalability compared to other candidates (see Table 1.1) [7], [8].

1.2 Phase-Change Memory as Main Memory

1.2.1 Opportunities of PCM-based System

PCM gains attention as the next-generation NVM, owing to its non-volatility, high-endurance, and scalability [9], [10], [11]. These characteristics allow new non-volatile memories to replace existing main memory or storage by adding a flexible new memory layer for the current computer architecture hierarchy. In recent years, software-defined memory has been announced to utilize NVM as high-speed storage or expanded main memory interchangeably [12], [13], [14], [15]. In particular, applications of in-memory databases require data to be persisted with lower latency; hence, a high-performance database can be developed by employing PCM as a non-volatile main memory [16], [17], [18], [19], [20], [21], [22], [23].

Furthermore, another attractive characteristic of PCM is its low standby power, as shown in Table 1.1. Unlike DRAM, PCM does not require (or rarely requires) refresh operations for data retention due to its non-volatility. Although PCM has a high dynamic power (i.e., read power and write power) compared to that of DRAM, the low standby power can be leveraged for a computation-intensive application or a platform having a long idle time duration (e.g., mobile devices). For example, the inference process of convolutional neural network (CNN) has extremely high data locality, thereby incurring a few accesses on the main memory. In such a case, PCM can be a suitable main memory device for achieving low power while hiding the longer latency with small data buffers.

In recent years, 3D-XPoint, which is originated from PCM [24], has been proposed as a gap filler between memory and conventional storage systems [25], [23]. These products have been tested in various environments for evaluating the performance and

exploring their suitable applications, such as scientific calculation, enterprise database, high-performance computing system, and etc. [26], [27], [28], [29], [30], [31], [32]. In conclusion, leveraging and enhancing PCM-related technology with currently available main memory devices is crucial to attaining low-latency and extra-scale memory-oriented computing systems in the future.

1.2.2 Challenges of PCM-based System

Despite its characteristics of non-volatility, latency, and high-endurance, which make PCM superior to NAND flash storage and DRAM, PCM is still not ready to be widely commercialized due to three major problems: lower cell reliability than DRAM, higher latency than DRAM, and lack of a well-established simulation platform.

First of all, there are several kinds of reliability issues in PCM, such as cell endurance [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], resistance drift [51], [52], [53], [54], [55], [56], [57], and read/write disturbance [58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69]. Write disturbance error (WDE) is one of the major problems, which delays its massive commercialization [59], [58], [60], [61], [62]. WDE is an interference problem on adjacent cells similar to row-hammer in DRAM [70], [71], [72], [73], [74], [75]. This problem must be addressed as the highest priority because it would be exacerbated as the technology scales down [60]. Additionally, applications of the in-memory database directly store data in NVM by utilizing cache-line flushes [23], [20], [19], [17], [18]. This kind of application would incur frequent write operations, thereby making cells vulnerable to WDEs. Therefore, mitigating the write disturbance notably with negligible overhead is requisite.

The second problem is a high latency for a write operation that requires no less

than 500 ns - 1000 ns, which is much slower than that of DRAM [76], [77], [78], [79]. The high latency can be compensated with the improved throughput. The high throughput can be achieved by increasing the size of a memory transaction (i.e., page size) rather than using the conventional 64 B, as explained in [10]. In [10], it is pointed out that a 512 B page 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 [80], [81], [82], [83], [84]. However, the conventional general-purpose processor typically adopts a 64 B 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 128 B) and a cache line size (i.e., 64 B) [85]. The redundant read operation in RMW for write access causes performance degradation and read disturbance errors. Therefore, it is necessary to enhance the performance of the PCM-based memory system having an RMW module.

Another problem is that there is no well-established PCM simulation platform for academic research purposes. Although Intel has manufactured several PCM-based products (e.g., Intel Optane DCPMM and Intel Optane SSD) [86], [27], Intel does not publish any well explained data-sheet for the product due to confidential issues, unlike DRAM or NAND storage devices. Therefore, state-of-art memory architecture simulators for PCM (e.g., NVMain [87], [88]) still lag behind the currently available industrial products. Even though the PCM has similar cell array structure as DRAM, the cell characteristics of PCM result in completely different memory controller architecture. For example, since PCM has limited cell endurance, address remapping is required with an additional subsystem that comprises an address translation mechanism. Furthermore, a larger transaction unit (i.e., larger than 64 B) is required for a PCM-based

memory system due to higher latency than that of DRAM; hence, a read-modify-write (RMW) module is also necessary to fill the gap of units between the traditional processor and the PCM-based memory system. Thus, a practical PCM controller simulator needs to takes these characteristics into consideration for more rigorous academic research purposes instead of merely modulating timing parameters in a simulator that is only composed of a command scheduler.

1.3 Dissertation Overview

This dissertation is to resolve challenging issues of PCM mentioned in Section 1.2.2 by developing the technologies of submodules in a PCM controller system, considering reliability, performance, and practicality. The following items summarize the key ideas of each contribution:

- **IMDB (Chapter 3)** leverages an SRAM-based table to record the data patterns and restores the WDE-vulnerable data on demand. The replacement policy dedicated to IMDB originally requires numerous read ports for an SRAM; however, a novel randomized selection can eliminate this overhead.

- **RMW in a PCM-based system (Chapter 4)** incurs significant performance overhead along with more read disturbance errors. This chapter provides a reliable boosting scheme that notably reduces such performance overhead by levering existing resources in a practical PCM controller system.

- **PCMCsim (Chpater 5)** is an all-inclusive simulator for a modern PCM controller, which incorporates all existing features that Intel publicly announces. The simulator also includes the most recent JEDEC DDR4 specification. Furthermore, the simulator provides a structured programming model and a contention behavioral model for future academic research.

Please note that each chapter listed above is generally composed of motivation, description of the scheme, evaluation, further discussion of the scheme, and the summary of the scheme. In addition, Chapter 2 explains necessary previous studies. Lastly, Chapter 6 concludes all contents of the dissertation.

Chapter 2

BACKGROUND AND PREVIOUS WORK

2.1 Phase-Change Memory

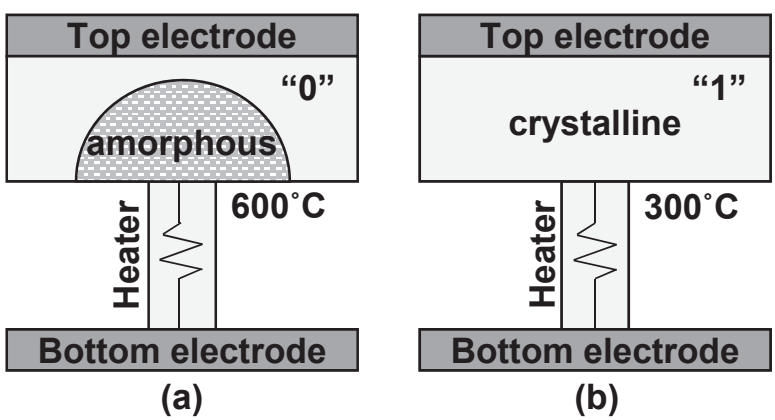


Figure 2.1: Different cell states of phase-change memory under different temperature. (a) amorphous state, (b) crystalline state.

PCM is a resistive memory device having two physical states, namely *amorphous* and *crystalline*, as shown in Figure 2.1. While the amorphous state (i.e., RESET) is achieved by heating the electrode on the cell above 600 degrees Celsius for a short

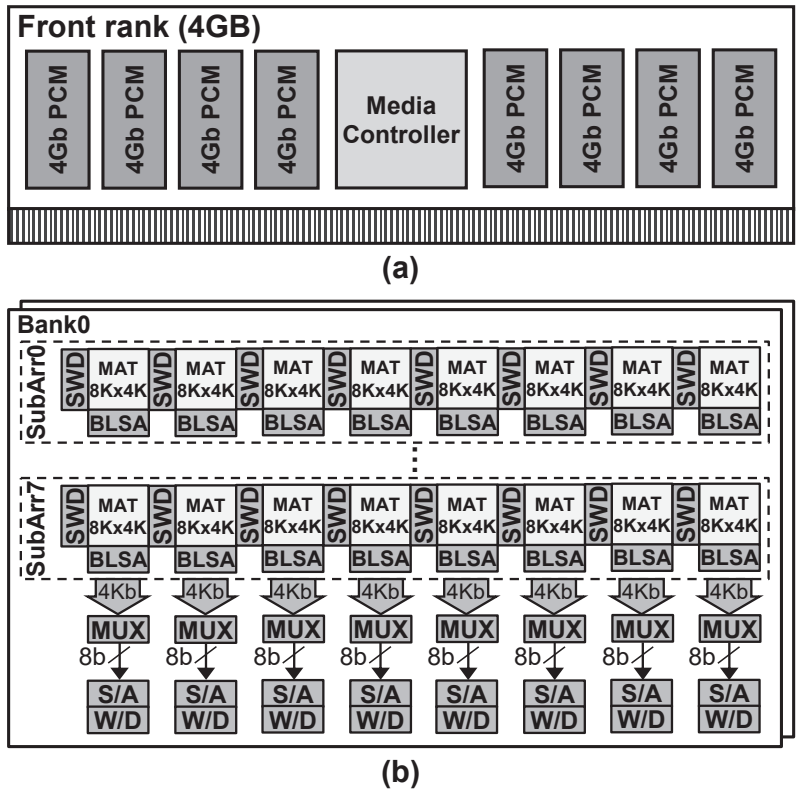


Figure 2.2: Illustration of PCM. (a) Front rank of a 8 GB PCM DIMM module, (b) architecture of a 4 Gb PCM device.

time, the crystalline state (i.e., SET) is achieved by supplying 300 degrees Celsius, followed by a longer quenching time than the amorphous state [77], [89], [90].

An 8 GB dual-rank DIMM module is illustrated in Figure 2.2 (a), in which Figure 2.2 (b) shows the detailed structure of a 4 Gb PCM device in the module. The device consists of eight subarrays, and each subarray is composed of eight MATs (8K wordlines and 4K bitlines for each MAT). First of all, main wordline drivers activate a subarray in each bank, and the row address is commonly fed into sub-wordline drivers (SWD) in the activated subarray for selecting a row that carries 4 Kb data. Subsequently, the selected 4 Kb data are sensed by bitline sense amplifiers (BLSA) and

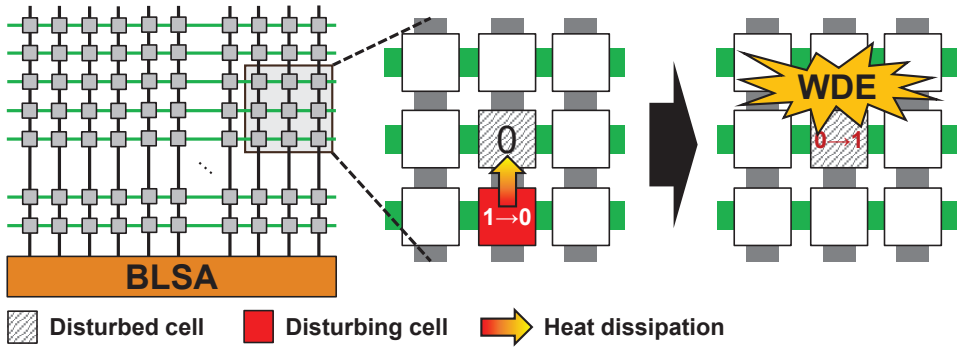


Figure 2.3: WDE in a PCM cell array. The hatched pattern is a disturbed cell.

transferred through global bitlines (shaded arrow in the figure). Each column multiplexer (MUX) obtains 4 Kb data from global bitlines and outputs 8-bit data to global sense amplifiers (S/A) using the column address. This output procedure means that eight consecutive bitlines comprise one column. Finally, 8-word data are transferred to the data bus in burst mode if eight data pins per device are assumed, and totally 64 B are carried out since eight devices are driven symmetrically by a single command. For a write operation, data on write drivers (W/D) are written back to the cell array with differential write [37]. Please note that the industry has presented that the PCM cells are organized such that they overlap with bitlines. Thus, WDEs mainly occur on adjacent materials patterned on a common bitline due to the simpler heat dissipation along bitlines [64].

2.2 Mitigation Schemes for Write Disturbance Errors

2.2.1 Write Disturbance Errors

Write disturbance error (WDE) is caused by the resistance shift from the amorphous state to the crystalline state [59], [58], [66], [69]. As shown in Figure 2.3, WDEs

occur on an idle cell adjacent to the cell under RESET operations [59], [58]. Since the intensity of current during a SET operation is nearly half of that during a RESET operation, an idle cell's temperature next to the programmed cell would be higher than those under SET (but lower than those cells under RESET). As a consequence, a phase transition may occur on that idle cell.

Awareness of the occurrence of WDEs is also crucial for modeling WDEs in a simulator. Instead of randomly triggering errors, the study in[67] shows that an amorphous cell gradually shifts to the crystalline state due to heat transfer to neighbors, thereby incurring WDEs. The study also explains that a cell can be programmed in different time frames, which means that WDEs can occur regardless of the idle time duration between consecutive writes. From this perspective, previous studies have reported that a WDE occurs when a cell experiences more than 5K-10K RESET pulses [62], where the number of pulses is referred to as the *WDE limitation number* in this study. In particular, this number assumes the worst case in which a WDE is triggered statically when the number of neighboring RESET pulses reaches the WDE limitation number. In this study, we assume that the WDE limitation number is **1K** (i.e., WDEs occur when the cell is exposed to 1K of 1-to-0 neighboring flips), according to the value reported by [62].

While WDEs are caused by heat dissipation on neighbors, rowhammer errors in DRAM occur when a row experiences more than 139K neighboring activations [72], [75]. Thus, the occurrence mechanism of WDEs seems to be similar to that of rowhammer in DRAM except for rationale in physics. However, schemes for mitigating rowhammer cannot be applied to PCM. A recent and representative study in [75] manages activation counters as a table and triggers refresh when the threshold is reached. Since the numbers of both activations and victim rows are bounded within

Figure 2.4: Step-by-step illustration of VnC. The yellow colored number represents the flipped bit exposed to WDE.

a refresh interval, the table size and threshold can be determined with mathematical deduction. Furthermore, the refresh is not obligatory in PCM; hence, one counter is allocated per PCM line, resulting in an impractical design option.

2.2.2 Verification and Correction

To reduce the number of WDEs to a manageable level, several mitigation schemes have been proposed previously. One of the representative and solid methods is verification-and-correction (VnC), which can completely eliminate WDEs with a simple approach [58]. As shown in Figure 2.4, VnC basically follows the process of ”read-write-read”:

1. Before writing the objective line, two neighbors of the objective line are read to

obtain the ”answer” of correct data.

2. Objective data is written while dissipating the heat on two neighbors. In this example, a bit of the upper line is disturbed.

3. Two neighbors of the objective line are read again, which are verified against the ”answer” obtained in step-1.

4. Since VnC confirms that an error has occurred in the upper line, step-1 to 3 of VnC are performed on the upper line.

According to the description above, VnC requires at least four read operations for guaranteeing data integrity. Furthermore, four read operations VnC must be strictly ordered by one write command. In conclusion, VnC significantly degrades the system performance (i.e., more than 50%).

2.2.3 Lazy Correction

Since VnC incurs considerable performance overhead, *lazy correction* has been presented for reducing such an overhead while ensuring the data integrity as well [58]. As shown in Figure 2.5, *lazy correction* is built on top of an additional chip, called the error-correction pointer (ECP) chip. An ECP chip is used for storing positions of worn-out cells [43], in which the locations of disturbed cells are also temporarily stored. As shown in Figure 2.5, red colors in the normal data field represent the bit positions having WDEs. The ECP field in the orange color records the bit positions of those errors. The data can be directly written to the device without VnC until entry overflow in the ECP chip. Consequently, the correction process of VnC can be deferred as late as possible until the correction entries become full. Still, four initial read operations of the

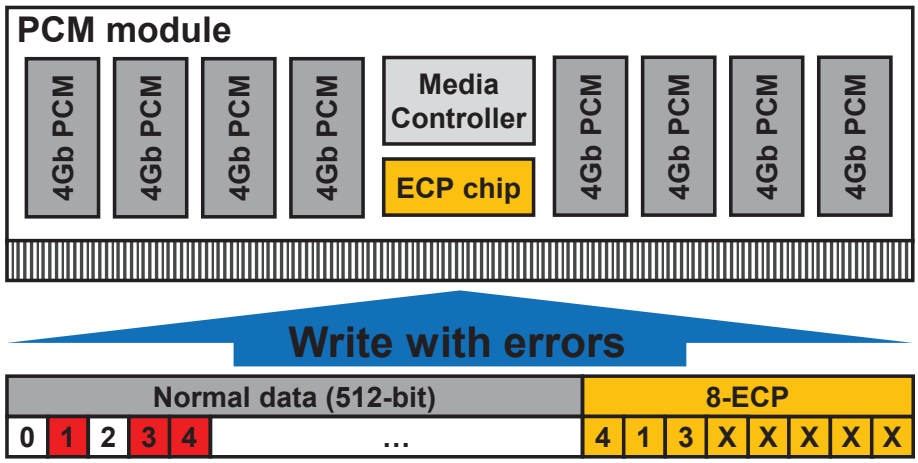


Figure 2.5: Illustration of lazy correction with eight error correction entries in the ECP chip. Red colors in the normal data field represent the bit positions having WDEs. The ECP field in the orange color records the bit positions of those errors.

original write command are unavoidable. Furthermore, cells in the ECP chip must be well insulated to guarantee no WDEs; hence, *lazy correction* inefficiently requires two different process technologies for manufacturers (i.e., a larger one for ECP devices and a normal one for normal devices). Therefore, *lazy correction* is less feasible to be adopted in the industry due to the limitation of the ECP chip.

2.2.4 Data Encoding-based Schemes

To reduce the high-cost four read operations incurred by VnC, previous studies have tried to utilize data encoding methods to reduce WDE-vulnerable patterns. Therefore, we can expect fewer or no VnC operations.

Data INsulation framework (DIN) [59] proposes a ”codebook” that encodes contiguous 0s in a compressed pattern to eliminate patterns vulnerable to WDEs. However, the encoded data need to be in the range of the length of the cache line (i.e., 512 bits);

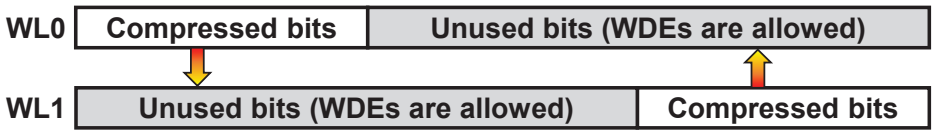


Figure 2.6: Illustration of ADAM. No harmful WDEs exist in this example because unused bits are free to be damaged.

otherwise, this approach must fall back on the VnC scheme, which incurs significant performance degradation in that case.

MinWD [61] encodes write data into multiple candidates with special shift operations. Subsequently, it elects the least aggressive form from all candidates, which is a final form that is written to the PCM device. However, this method requires additional bits as an indicator of the shift operation, incurring at least 12.5% storage overhead (i.e., 64-bit/512-bit=12.5%).

Architecture for Write DisturbAnce Mitigation (ADAM) [60] observes that frequent pattern compression (FPC) can highly compress the cache line data in most CPU applications. This work compresses all cache lines with on-the-fly FPC and aligns the line to the right and left alternately; hence, the number of valid bits on adjacent rows is lowered, as shown in Figure fig-ADAM. However, encoding schemes strongly depend on the data patterns of the applications, leading to limited mitigation performance (i.e., 53% WDE reduction).

In conclusion, data encoding-based schemes can be leveraged without VnC. However, decoupling from VnC leads to lower system reliability, thereby requiring the use of error-correction coding (ECC).

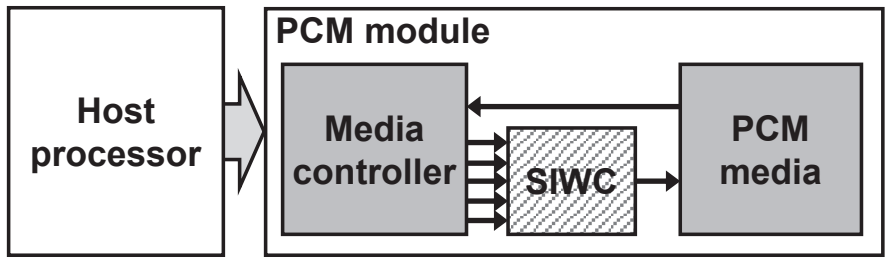


Figure 2.7: Illustration of SIWC. Intensive writes are buffered by SIWC, yielding fewer WDEs.

2.2.5 Sparse-Insertion Write Cache

Volatile data caches (e.g., SRAM) can enhance both the throughput and reliability of a system by storing frequently updated data. *Sparse Insertion Write Cache (SIWC)* leverages a write cache that inserts data probabilistically and absorbs bit flips [62], as shown in Figure 2.7. Since data vulnerable to WDEs would be stored in the write cache, the victims of WDEs become safe. However, it introduces several megabytes of volatile memory to obtain a high hit ratio. Furthermore, even if the write cache is embedded in the memory module, the supercapacitor required for data flush upon system failure has to be expanded as the volatile region enlarges. Typically, a commercial non-volatile dual in-line memory module (NVDIMM) guarantees that the volatile data must be flushed within 100us [91]. Therefore, the cache capacity must be reduced to an affordable value. Moreover, it is reported that WDEs likely occur when cells are exposed to RESET for specific times [62], but it does not utilize such a feature, which is key for highly reducing WDEs.

2.3 Performance Enhancement for Read-Modify-Write

2.3.1 Traditional Read-Modify-Write

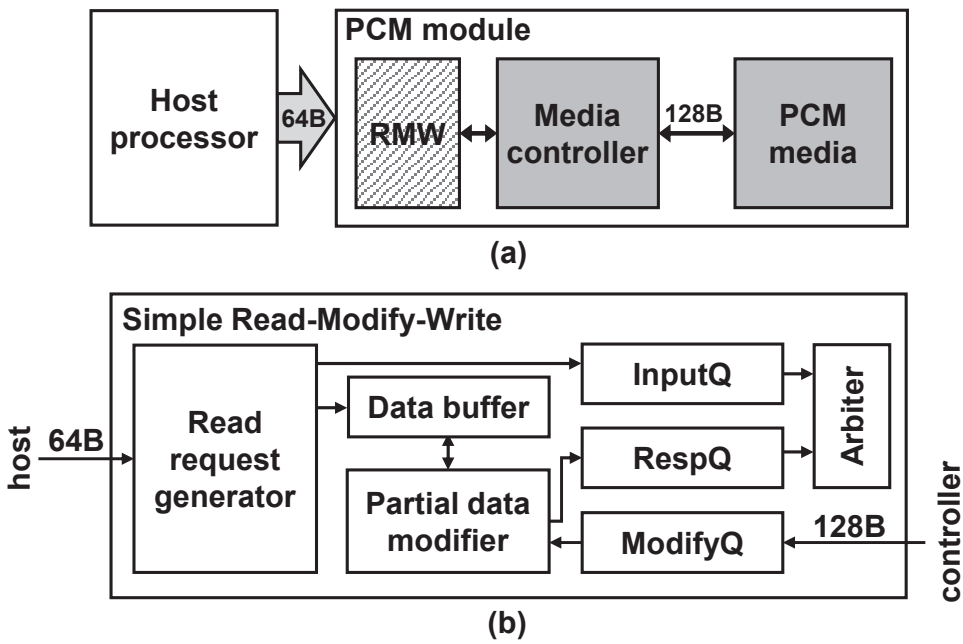


Figure 2.8: An RMW module in a PCM-based system. (a) Overview of an RMW-enabled PCM module, (b) detailed view of the simple RMW module.

As explained in Section 1.2.2, a read-modify-write (RMW) module is required in a PCM-based system to fill the transaction unit gap between the conventional CPU (64 B) and the PCM-based memory system ($\geq$ 128 B). In the hatched region in Figure 2.8 (a), the RMW module 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 page size) of the PCM for each access, where each data block has a length of 64 B. 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 page-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 precharge, activation, read, and write.

The baseline model of the RMW is depicted in Figure 2.8 (b) with details. There are three bits of flags for RMW operation, *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".
   - If the type is *WRITE*, *WAS WRITE* is set to "1" to "remember" its original command as *WRITE*.

   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 page 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.

2.3.2 Write Coalescing for RMW

The redundant read operation in RMW for write access causes performance degradation. As no RMW scheme for a PCM has been studied so far, schemes proposed for other devices (e.g., DRAM) are carefully investigated. In general, write coalescing is one of the representative operations for an RMW module in a DRAM-based system [92], [93]. Write coalescing targets on graphics applications or GPUs. These applications directly access DRAM to update small-sized pixel values (e.g., 32-bit), thereby incurring RMW and updating 64 B data finally. Since these applications update consecutively aligned pixels, write coalescing can effectively merge several 32-bit write

commands into a few 64 B write commands. For example, updating 16 pixels incurs 16 read commands and 16 write commands originally, whereas only 1 write command is necessary if the write coalescing is applied. Therefore, the write coalescing can significantly enhance system performance if graphics applications dominate the system.

In contrast, normal applications have few consecutive write operations. Furthermore, 64 B write commands are more frequent in normal applications. Thus, lowering the performance overhead of RMW is not crucial in a DRAM-based system in general if there is no graphics application running on the system. However, on the other hand, there is a page size gap in a PCM-based computing system, as explained in Section 1.2.2. Therefore, the performance degradation due to RMW in a PCM-based system becomes notable.

2.4 Architecture Simulators for PCM

2.4.1 NVMain

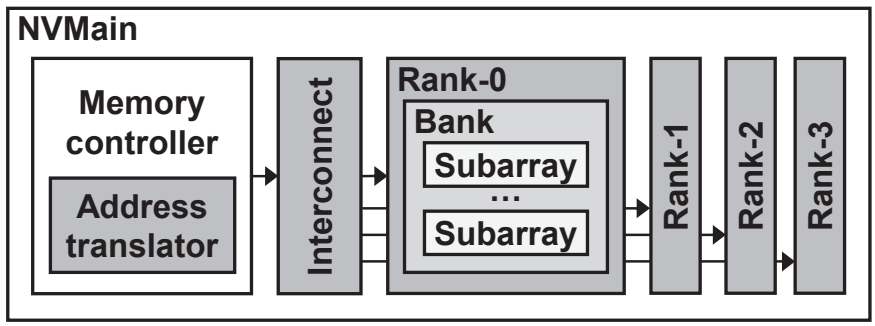


Figure 2.9: Architecture of NVMain that simulates a 4-rank NVM system.

NVMain, specifically NVMain 2.0, is an event-driven NVM simulator that is widely used in PCM-related research [18], [40], [94]. As shown in Figure 2.9, NVMain consists of the basic memory controller architecture, interconnect module, and device objects (e.g., rank, bank, subarray, and etc.). In particular, NVMain describes the device objects down to cell-level, in which the endurance model of NVMs is included as well. Furthermore, different from the first version in [87], NVMain 2.0 [88] introduces advanced address translators and memory object hooks. The former enables the remapping of the logical address to the physical device address by predefining the address field in the configuration file. The latter facilitates the programmers to snoop the in-flight memory requests and change them dynamically during run-time. However, NVMain only supports JEDEC DDR3 specifications. It is noteworthy that DDR4 becomes populated in recent memory-related research [95]. Furthermore, self-refresh is one of the important features not only in DRAM but also PCM because data retention issues also exist in PCM; however, self-refresh is not implemented in NVMain (i.e.,

only parameters of tXS and tXSDLL are provided without functional behaviors).

2.4.2 Ramulator

Ramulator is a representative cycle-level DRAM simulator that supports DDR4 [96]. Although Ramulator is a cycle-level simulator, it leverages a curiously recurring template pattern (CRTP) feature in C++ that can significantly reduce the lookup overhead of virtualized functions, yielding fast simulation time than other existing memory architecture simulators. Ramulator has validated its timing correctness against Micron's DDR3 RTL model written in Verilog. However, Ramulator shows low flexibility concerning parameter configurations. In order to reduce the simulation time as much as possible, Ramulator embeds device parameters (e.g., timing or channel width) in the dedicated data structures written in the source code. That is, if a user would like to analyze the behavior of a new memory architecture that does not exist in the market with various sensitivity analyses, all desired configurations must be first written in the source code, followed by re-compilation to different binary files for different configurations. Consequently, lower productivity is inevitable, considering plenty of experiments in industry or academic research.

2.4.3 DRAMsim3

DRAMSim2 [97] has been widely utilized in various memory-related research, even in NVMs [98], [19], [23]. However, DRAMSim2 only supports DDR3 as the latest standard, which shows low feasibility for both industry and academic research. Furthermore, other simulators (e.g., Ramulator) have low accessibility due to their complexity. To overcome this problem, DRAMsim3 [99] is presented in recent years (i.e., 2020). Different from DRAMSim2, DRAMsim3 introduces various important features

for recent DRAM devices (i.e., DDR4):

- **Bankgroup timings** is an essential feature in DDR4. While a DDR4-based DRAM device internally has the same prefetch size as DDR3 (i.e., 8n), it can maximally yield the throughput of 16n prefetch size by accessing different bankgroups at a time [100], [101]. However, the constraints of accessing the same bankgroups are slightly longer due to the contention on the local sense amplifier. In DRAMsim3, bankgroup timings are introduced for simulating DDR4-based devices.

- **Flexible address mapping** becomes important recently due to some security issues (i.e., Rowhammer [74], [73]). Therefore, secure address mapping is required to prevent the exploitation of such malicious attacks. In DRAMsim3, an arbitrary address mapping module is implemented to allow researchers to explore such kinds of issues.

- **Bank-level refresh** is common in recent DRAM devices. As the density of DRAM devices becomes higher, more rows must be refreshed with one refresh command. However, refreshing more rows across all banks (i.e., all-bank refresh) yields higher instant power, leading to unexpected power-down in a computing system. Thus, bank-level refresh features, such as per-bank refresh, are introduced in DRAMsim3.

Still, DRAMsim3 is a cycle-level memory simulator without necessary features (i.e., address remapping subsystem or read-modify-write process) for a PCM controller, hindering the rapid growth of PCM-related research.

Chapter 3

IN-MODULE DISTURBANCE BARRIER

In the first dissertation, we propose write disturbance error (WDE) mitigation scheme in a PCM module. WDE appears as a serious reliability problem preventing PCM from general commercialization. WDE occurs on the neighboring cells of a written cell due to heat dissipation. Previous studies for the prevention of WDEs are based on the write cache or VnC while they often suffer from significant area overhead and performance degradation. Therefore, an on-demand correction is required to minimize the performance overhead. In this chapter, an in-module disturbance barrier (IMDB) mitigating WDEs is proposed. IMDB includes two sets of SRAMs into two levels and evicts entries with a policy that leverages the characteristics of WDE. In this work, the comparator dedicated to the replacement policy requires significant hardware resources and latency. Thus, an approximate comparator is designed to reduce the area and latency considerably. Furthermore, the exploration of architecture

The short version of this chapter is originally presented in Design Automation Conference (DAC), work-in-progress (WIP) session, 2020 [102].

Table 3.1: Performance of randomized VnC

Probabilities of VnC			**WDE reduction**	**Speedup**
both rows	upper row	lower row		
0%	50%	50%	23%	57%
75%	12.5%	12.5%	30%	18%
80%	10%	10%	36%	17%
90%	5%	5%	36%	15%
95%	2.5%	2.5%	43%	15%
99%	0.5%	0.5%	46%	14%

parameters is conducted to obtain cost-effective design. The proposed work significantly reduces WDEs without a noticeable speed degradation and additional energy consumption compared to previous methods.

3.1 Motivation

The necessity of reducing the performance overhead of VnC. VnC, the most common solution to WDEs, triggers read operations to read two neighboring data before the objective data is updated. Subsequently, two neighbors are read again after the write operation for verification. Finally, VnC is performed iteratively if WDEs occur on the neighbors. As a result, the performance is degraded markedly with these iterative read operations. A naive approach to reduce the number of VnC operations is to perform VnC randomly. Specifically, one of the three operations is executed randomly (see the first three columns in Table 3.1). As shown in Table 3.1, random VnC yields a 14% speedup compared to normal VnC and a WDE reduction rate of 46% compared to a raw machine (i.e., no WDE mitigation scheme). This occurs because PCM does not

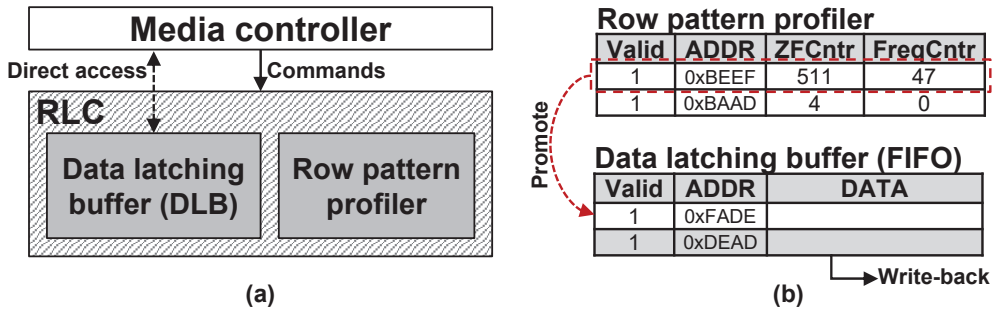


Figure 3.1: Illustration of RLC. (a) overview, (b) mechanism of data structures.

require a refresh operation by default (or an infrequent refresh compared to DRAM), causing cells scarcely to be restored. In contrast, a high speedup (i.e., 57%) is attainable at the expense of reliability. Moreover, the operations of VnC (i.e., pre-write read, write, and post-write read) are strictly ordered; hence, the speedup is not notable even when a probabilistic approach is applied. As a result, a new on-demand approach is required because even the randomized approach shows low mitigation performance and a low speedup in PCM-based systems.

Limitation of VnC-based schemes. As a naive approach for reducing the frequency of VnC, the verified data verified can be stored in a reliable memory region, such as an SRAM. As shown in Figure 3.1, a row-latching VnC (RLC) can be an intuitive approach to address this problem. RLC consists of a data latching buffer (DLB) and a row pattern profiler. Both data structures are implemented with SRAM. The former caches data in a first-in-first-out (FIFO) manner, whereas the latter records the number of 1-to-0 bit flips that easily causes WDEs. When a write command is ready to issue verification read commands for VnC, the command first directly access DLB for checking the existence of verification data (i.e., neighbors' data). If the data exists in DLB, the verification read for that neighbor is skipped. Moreover, if the data of the write command exists in DLB, it is directly updated. Therefore, RLC can reduce

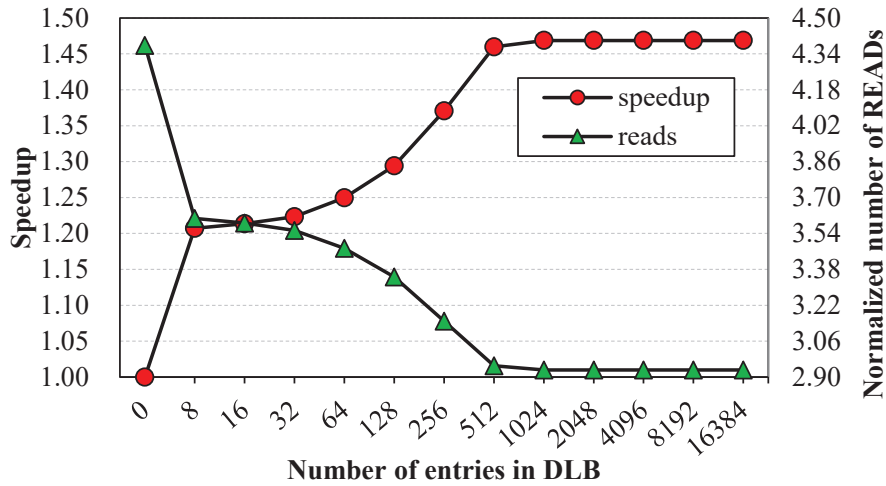


Figure 3.2: Speedup and the number of read commands compared to VnC.

the number of read commands significantly. Please note that data in DLB is promoted from the entries in the row pattern profiler. Figure 3.2 shows the speedup and the number read commands concerning different numbers of entries in DLB. As shown in the figure, the number of reading commands does not reduce from 1024-entry due to the strictly ordered commands by VnC.

The necessity of reducing the cache burden. Cache-based schemes mitigate WDEs by temporarily storing write data into dedicated SRAM. Although a cache-based scheme (i.e., *SIWC* [62]) can significantly reduce the number of WDEs on PCM compared to those in previous studies, this strategy requires high-capacity SRAM because it indiscriminately caches write data. Furthermore, data adjacent to cached addresses remain vulnerable to WDEs. To overcome these challenges, it is necessary to store the data that likely incur WDEs (i.e., WDE aggressors) and restore cells adjacent to these aggressors. Fortunately, WDE aggressors can be predicted with the WDE limitation number; hence, time-consuming VnC operations are unnecessary.

In conclusion, this dissertation proposes a comprehensive approach that determin-

istically restores cells that are seriously vulnerable to WDEs on demand and stores only the data of WDE aggressors in a small-sized cache to prevent upcoming WDEs in a PCM module. Thus, it is one of the industry-friendly approaches.

3.2 IMDB: In Module-Disturbance Barrier

3.2.1 Architectural Overview

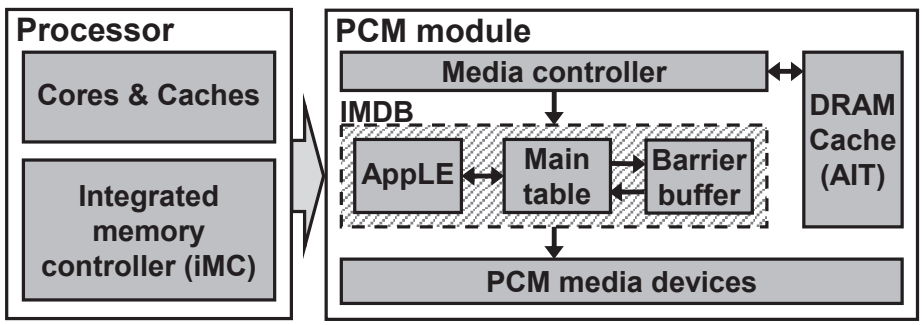


Figure 3.3: Architectural overview of IMDB.

Figure 3.3 depicts the overall architecture, where NVM commands are dispatched from the integrated memory controller (iMC) in the host. For the PCM module, the media controller generates micro-commands and schedules commands to available banks in the media devices. A DRAM cache is only used for storing address indirection table (AIT) [91], [103]. The proposed module, IMDB, is located between the media controller and media devices for the prevention of WDEs.

As shown in Figure 3.3, IMDB consists of a main table, a barrier buffer, and AppLE. Firstly, the main table manages the addresses of WDE aggressors. The number of 1-to-0 bit flips is calculated and accumulated in the table when a write address hits in the table. Otherwise, the dedicated replacement policy supported by AppLE, which reduces the overhead incurred by multi-port SRAM, selects a victim entry within the table and replaces it with the new address. When the number of bit flips on the aggressor exceeds the pre-defined threshold, IMDB generates *rewrite commands* for data that are adjacent to the aggressor. As explained in Section 2.2.1, an idle cell in amorphous state

(i.e., RESET) gradually shifts to crystalline state if it is exposed to high-temperature several times. Then, a WDE happens when this cell completely turns into crystalline state. Therefore, the rewrite command is introduced and used for restoring such partially shifted cells back to amorphous states before the occurrences of WDEs. Subsequently, IMDB migrates the information from the main table to the barrier buffer that comprises a few data entries, reducing WDEs further. Even though the barrier buffer's entry has longer data than that of the main table, the barrier buffer costs less SRAM capacity because it manages much fewer entries. Figure 3.3 shows the swapping mechanism between two tables, by which WDE aggressors are managed as long as possible within IMDB.

It is noteworthy that the architecture in Figure 3.3 is built on top of a practical PCM product. In recent years, products attempt to support expanded memory with special drive technology [14], [27]. Furthermore, some products are manufactured with the NVDIMM, which can be operated as main memory[86]. In particular, the architecture in this study stems from the latter. Therefore, it is reasonable to design IMDB based on such a baseline architecture in this study.

3.2.2 Implementation of Data Structures

Figure 3.4 shows the detailed architecture of IMDB, where each plane is allocated for every PCM bank; hence, all IMDB planes operate concurrently at the bank level without contention issues. An IMDB plane consists of two tables, namely a *main table* and a *barrier buffer*, where the following subsections describe implementations of each table in detail.

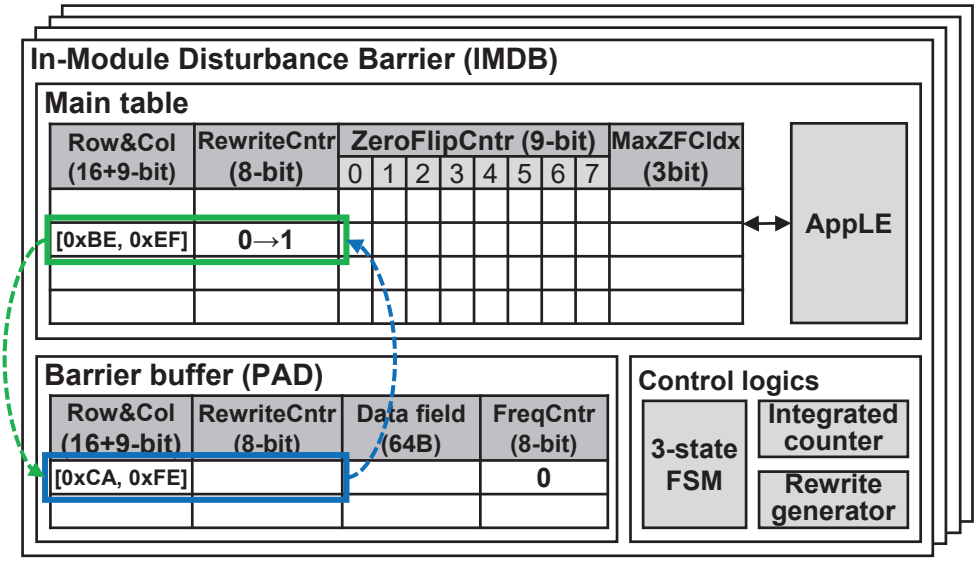


Figure 3.4: Implementation of four IMDB planes. Each IMDB plane is assigned to each PCM bank operation.

Main Table

The main table is implemented with a set of SRAMs, where the entry is updated by control logics. In particular, four fields exist in the table for estimating the degree of WDE of a write address:

- *Row & Col*: It contains the row and column addresses in a bank that are currently being managed.
- *ZeroFlipCntr*: Eight sub-counters are in the field, each of which counts the number of bit flips from 1 to 0 and manages one 64-bit word in a 64 B cache line.
- *MaxZFCIdx*: It indicates the index of sub-counter of *ZeroFlipCntr* holding the maximum value. It is updated in control logics after reading an entry. It is used for comparing the maximum value of the *ZeroFlipCntr* with the threshold value

for rewrite operations.

- *RewriteCntr*: An 8-bit counter represents the frequency of rewrite operations on the address of *Row & Col*.

A per-bank IMDB plane is assigned to each bank; hence, bank parallelism is ensured to lower the contention on IMDB. Furthermore, because one command is allowed to IMDB at a time, no serialized queue is necessary for IMDB, preventing resource redundancy. The command is handled by a 3-state finite state machine (i.e., IDLE, HIT, MISS) in control logics, where the varying latency due to multiple cases is factored in the simulator. After a command is inserted, IMDB operates in two different ways whether the address is found in the table or not:

- If the address is found in the main table, the state transits to HIT. Meanwhile, two data, i.e., the new write-data and the previously written data already read in the controller, are passed to control logics. Subsequently, the number of 1-to-0 bit flips is counted by integrated counters (see Section 3.3.2) and accumulated to the corresponding *ZeroFlipCntr*. When the maximum value of *ZeroFlipCntr* surpasses the predefined *threshold*, two rewrites on adjacent wordlines are generated and sent to the write queue in the media controller. Accordingly, the value of *RewriteCntr* increases. As system reliability is critical, the highest priority is conferred to the rewrite request.

- If an address is not found in the main table, an insertion is required while converting the state to MISS. The probabilistic insertion method is leveraged in this study, where infrequent accesses are filtered out with probability p to reduce evictions from the SRAM. When insertion is required, our proposed replace-

ment policy determines the victim (explained in Section 3.3), and thereby the new address can replace the victim entry.

In the proposed design, two parameters, (1) the threshold of generating rewrite commands and (2) the probability p, are necessary. First of all, we decide the threshold of generating rewrite commands in the main table as "*WDE limitation number/2-1*" because two rows can disturb a row. Thus, if we assume WDE limitation number of 1K, as in[62], the threshold becomes 511, making the bit width of each *ZeroFlipCntr* as 9. In order to justify the formula of the threshold value, we have evaluated the performance with different threshold values. Compared with the calculated value from the formula (i.e., 511), our evaluation indicates that threshold values of 1023, 2047, and 4095 increase the number of WDEs by 14.9×, 64.4×, and 142×, respectively. This is because the rewrite generation is triggered later than the occurrence of WDEs. Besides, speed and energy vary by 0.01% because the rewrite operation occupies less than 0.1% of all write operations. Therefore, overly augmenting the threshold will degrade the WDE mitigation performance.

The other parameter, p, indicates the probability of inserting a new missed address into the main table. Increasing the probability incurs more frequent entry replacement in the table detecting WDE aggressor, losing the opportunity to rewrite the victims of WDEs. In contrast, decreasing the probability makes "long-term" attacks lose the chance to be in the table. As shown in Figure 3.5, our experiments regarding different insertion probabilities show that p=1/128 yields the least WDEs; hence, we select p=**1/128**.

As shown in Figure 3.4, the main table employs two types of SRAMs. First, a dual-port content-addressable SRAM (CAM) is allocated as *Row & Col* fields. Second, a multi-port SRAM, consisting of *ZeroFlipCntr*, *MaxZFCIdx*, and *RewriteCntr*,

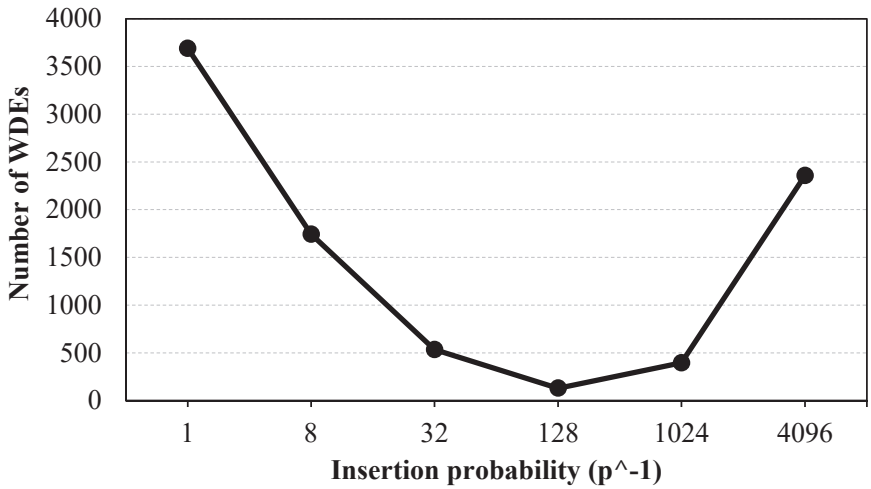


Figure 3.5: Absolute number of WDEs regarding different insertion probabilities.

has multiple read ports for obtaining all entry contents at once to apply the proposed replacement policy (see Section 3.3.1). However, since the use of multi-port SRAMs causes significant overhead, we propose *AppLE*, which enables the replacement policy with a DPSRAM without speed degradation (see Section 3.3.2).

Barrier Buffer

The barrier buffer is leveraged to store the data with frequent 1-to-0 bit flips. For a read request, the barrier buffer is capable of serving commands directly. For a write command, if the address hits on the barrier buffer, the data are updated in the barrier buffer directly. Otherwise (i.e., if an address only hits on the main table), the normal operation of the main table is performed, as explained in the previous subsection.

As shown in Figure 3.4, the green-boxed entry in the main table is the data frequently exposed to 1-to-0 flips. It is invalidated and *promoted* to the barrier buffer when *RewriteCntr* updates (i.e., rewrite occurs in the main table). The barrier buffer inherits the address and *RewriteCntr* information from the main table. If the barrier

buffer is not full, the promoted entry can be directly placed in the barrier buffer. The promoted entry replaces the least frequently used (LFU) entry that is bounded by the blue box in the figure; hence, *FreqCntr* is required for the replacement policy, as in [104]. The LFU entry data are sent back to the media controller for writing back the dirty data, and this information is *demoted* to the main table. Because the demoted addresses have been WDE aggressors before, the number of rewrites is reserved in *RewriteCntr*. *RewriteCntr* provides historical information with which to obtain a reasonable victim candidate in the main table (explained in Section 3.3.1). It should be noted that the 8-bit of *RewriteCntr* is a generously selected bit width to prevent overflow based on our experiments.

To implement the barrier buffer, a dual-port CAM-based SRAM and a dual-port SRAM are employed for *Row & Col* and *data & RewriteCntr & FreqCntr*, respectively. The energy consumption is negligible because only a small number of entries in the barrier buffer is enough to provide high WDE mitigation performance, as shown in Section 3.5.7. The sensitivity analysis of the number of entries would be shown in Section 3.5.5.

In terms of speed degradation, although the swapping mechanism (i.e., promotion & demotion) incurs additional latency, this latency is negligible. In detail, two cycles are required for reading contents from the main table and barrier buffer sequentially. Then, one cycle is necessary for writing swapped contents back to the tables again. In summary, a total of three cycles of latency are required for swapping; hence, the proposed method would not incur noticeable speed degradation (see the detailed results in Section 3.5.7).

The barrier buffer function seems to be similar to that of the previous caching scheme (i.e., SIWC [62]); however, the barrier buffer is different from the previous ap-

proach that cannot classify vulnerable patterns. To tackle this problem, our proposed method appends a newly structured preprocessor (i.e., main table) that can detect WDE aggressors and predict addresses vulnerable to WDEs. In particular, the detected aggressors are promoted to a much smaller data cache (i.e., barrier buffer) than that of the previous scheme. Thus, the proposed method can effectively reduce WDEs with lower hardware complexity compared to previous caching schemes.

It is noteworthy that the SRAM is a more suitable media acting as the barrier buffer rather than DRAM. If the on-die DRAM acts as the barrier buffer instead of using SRAM, swapping contents between the main table and the barrier buffer requires several cycles to transfer the contents between the two types of media. In contrast, the proposed method requires fewer cycles. When an address in the main table is detected as a WDE aggressor, the rewrite command is sent to the media controller instantly. Subsequently, the address is ready for promotion to the barrier buffer, and an entry for demotion is selected in the barrier buffer. Finally, the contents of the two entries are swapped into each buffer simultaneously. Given that this three-step process incurs two sequential reads and two concurrent writes on SRAMs, 3 cycles of latency are required for swapping. The proposed method would not incur noticeable speed degradation, as shown in Section 3.5.7.

3.2.3 Modification of Media Controller

The media controller is modified slightly to support IMDB in two aspects. First, acquiring previously written data is necessary to count bit flips. Thus, a *pre-write read operation* is performed ahead of a write command for this purpose. To temporarily store the previous data, the controller holds one more data buffer to carry the old data and introduces additional bits for distinguishing prepared commands from unprepared

ones. The pre-write read request has a higher priority than write requests but a lower priority than normal read requests because write requests in the controller mainly drain when the queue is full. Second, a merge operation is introduced, by which the rewrite command can coalesce with a same-address write command. It is noteworthy that a rewrite operation entirely writes all bits of data. Thus, the excessive number of rewrites may incur cascaded WDEs on neighbor data lines, whereas the merge operation can address this issue.

3.3 Replacement Policy

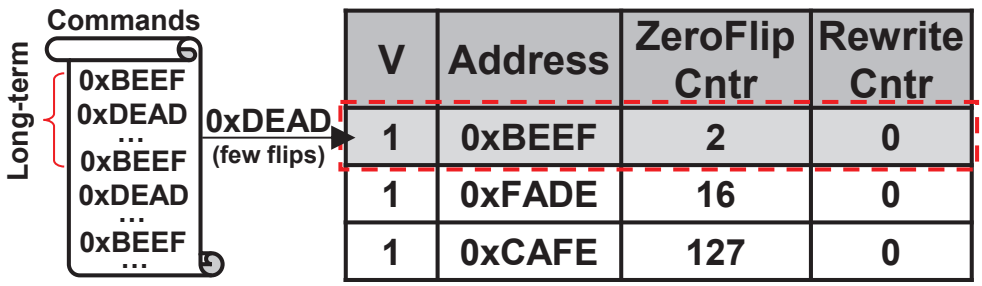


Figure 3.6: A toy example showing malicious attacks. 0xDEAD evicts insufficiently baked 0xBEEF, which is vulnerable to WDEs with gradual 1-to-0 bit flips.

This section describes implementation details of the replacement policy for the main table. Furthermore, AppLE is proposed to address the issue of multi-port SRAM incurred by the policy.

3.3.1 Replacement Policy for IMDB

A replacement (or eviction) policy is required in the main table based on the knowledge of WDEs. One of the representative indicators showing the vulnerability of WDE is the number of 1-to-0 bit flips, which is accumulated in *ZeroFlipCntr*. The other one is *RewriteCntr*, which records the historical occurrences of rewrite operations on the victims of WDEs. Therefore, we exploit *ZeroFlipCntr* and *RewriteCntr* to define the replacement policy.

When the input command requests a new entry in the main table, the policy is ready to select the victim entry. The victim candidate is defined as a less urgent aggressor, thereby selecting the minimum value of *ZeroFlipCntr*. However, more than two candidates may exist if the table has entries with the same values of *ZeroFlipCntr*. Since

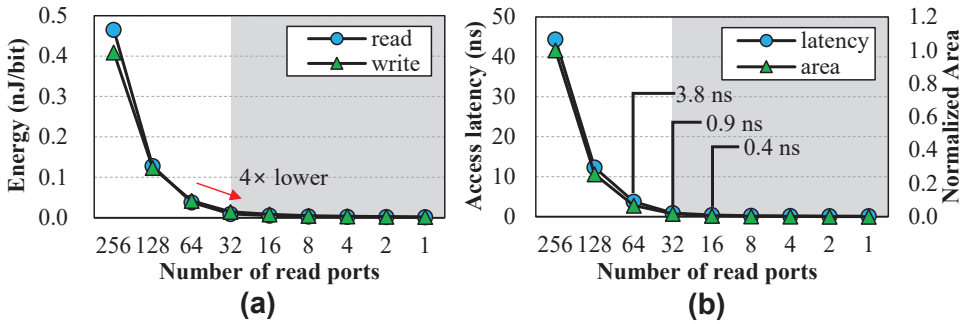


Figure 3.7: Energy, latency, and area of a 256-entry SRAM having multiple read ports, which is extracted from CACTI [105]. (a) energy, (b) latency and area.

the aggressiveness of WDEs varies with historical information (i.e., *RewriteCntr*), the entry containing the minimum of *RewriteCntr* is finally selected as the replaced entry.

To prevent ”cold-start” that incurs early eviction from the table, this study confers *prior knowledge* to prevent the entry from early eviction. Since the policy prioritizes the present vulnerability using *ZeroFlipCntr*, the recently inserted but insufficiently ”baked” entry can easily be evicted from the main table. Although *RewriteCntr* contains the historical information, it would be useless if the entry is newly inserted and evicted right away unluckily (see example in Figure 3.6). To tackle this problem, the prior knowledge, which is simply defined as the number of zeros in each data block, is initialized *ZeroFlipCntr* in the main table.

It is noteworthy that a module, namely *integrated counter*, is required to perform the above processes. The integrated counter provides mainly two functions. First, it counts the number of 0s of newly inserted data, which is then directly used as *prior knowledge* of *ZeroFlipCntr*. Second, it counts the number of 1-to-0 bit flips of the accessed address in the table, where the counted value is added to the *ZeroFlipCntr*. As a result, the integrated counter is implemented as Figure 3.8, where eight counter

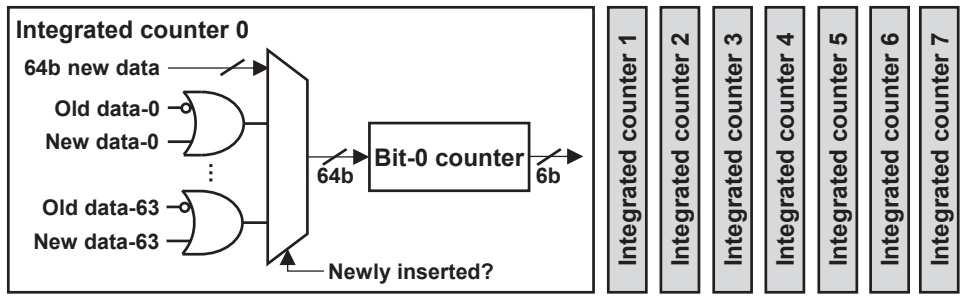


Figure 3.8: Integrated counters for eight *ZeroFlipCntr*s.

blocks are required to count each 64-bit word in 64 B concurrently. Our hardware synthesis results show that the area of all integrated counter blocks accounts for 0.15% of the typical DRAM memory controller, i.e., *PARDIS* [106]. The complexity of the PCM controller is obviously higher than that of the DRAM controller because the PCM controller requires more submodules (e.g., the wear leveling and the interface of DRAM AIT). Therefore, the integrated counter occupies only a small area in the PCM module.

3.3.2 Approximate Lowest Number Estimator

The eviction policy requires the numbers of read ports and entries on the main table to be equal, which increases latency, area, and energy overheads. If a 256-entry main table is assumed, 255 tree-structured dual-input comparators are necessary for latency minimization (i.e., 8 cycles). However, our evaluation results in Figures 3.7 (a) and (b) indicate that a large number of read ports on an SRAM can significantly increase both energy, latency, and area. As a result, an SRAM with 256 read ports is an infeasible implementation considering such aspects of overhead.

To reduce such aspects of overheads, this study introduces a sampling-based comparator, namely AppLE. The basic concept of AppLE is to bind a few entries (e.g., 8

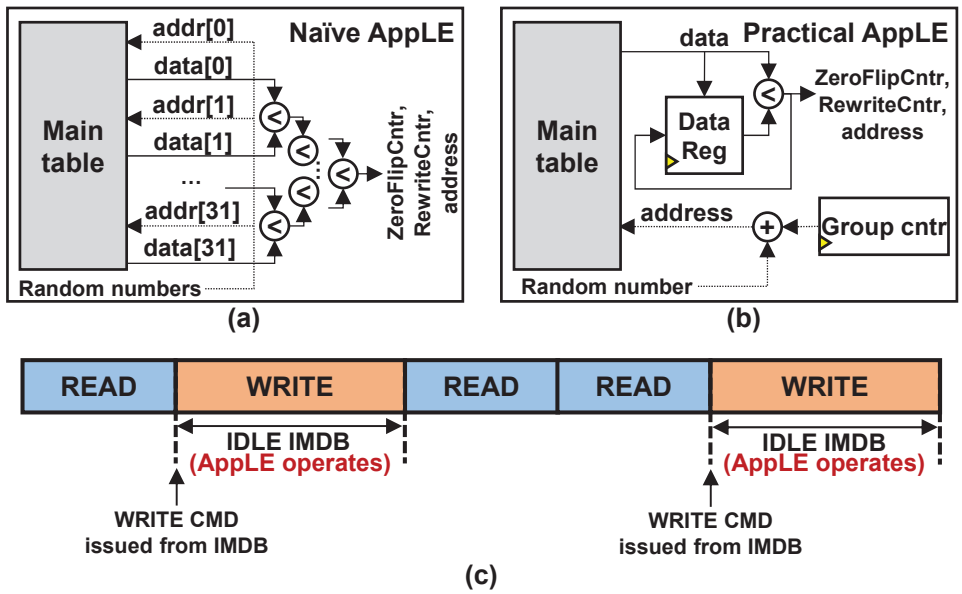


Figure 3.9: Implementations of AppLE. (a) a naive approach, (b) a practical approach, (c) timeline of IDLE state in IMDB.

entries) as a group, yielding 32 groups. Subsequently, each group randomly generates a number ranging from 0 to 7. The generated number is added to the value of *group-index*$\times 8$, which becomes the main table's input address for obtaining a *sampled entry*. Consequently, the victim candidate is selected among sampled entries according to replacement policy.

A naive and *impractical* approach to implementing AppLE is to treat each group as a read port, as shown in Figure 3.9 (a). According to the latency results in Figure 3.7 (a), since the typical I/O frequency of DDR4 is around 800 MHz [107], the maximum target number of read ports is set to 32. Still, the area of a 32-port SRAM is 105$\times$ larger than that of a single read port SRAM. Moreover, SRAMs consisting of dozens of read ports are unusual in terms of industrial manufacturing. Therefore, regarding the number of groups as the number of read ports is *infeasible* to implement AppLE.

To resolve this problem, this study proposes a *practical* design with DPSRAM (i.e., one read and one write), as shown in Figure 3.8. It treats the number of groups as the number of cycles; hence, sampled entries can be obtained from one read port by incrementing the *group counter* over multiple cycles. However, the challenge of this approach is the latency incurred by multiple read operations to obtain sampled entries from the main table. Nevertheless, this challenge can be addressed by hiding the latency. As shown in Figure 3.9 (c), IMDB becomes idle after issuing a write command (explained in Section 3.2.2). The idle state after dispatching a write command maintains for 150 ns (i.e., 120 cycles at 800MHz), which is the write latency of a modern PCM device [59]. Therefore, the latency of AppLE can overlap with the idle state of IMDB. Because the worst case of the replacement policy is observing all *ZeroFlipCntrs* and *ReWriteCntrs* sequentially, the maximum number of groups for implementing AppLE without additional latency should be 32. Consequently, AppLE only requires one comparator, a data register for storing the temporal results, and a register for counting group index. Since the group size determines the randomness and mitigation performance, we present the sensitivity analysis in Section 3.5.6.

3.4 Putting All Together: Case Studies

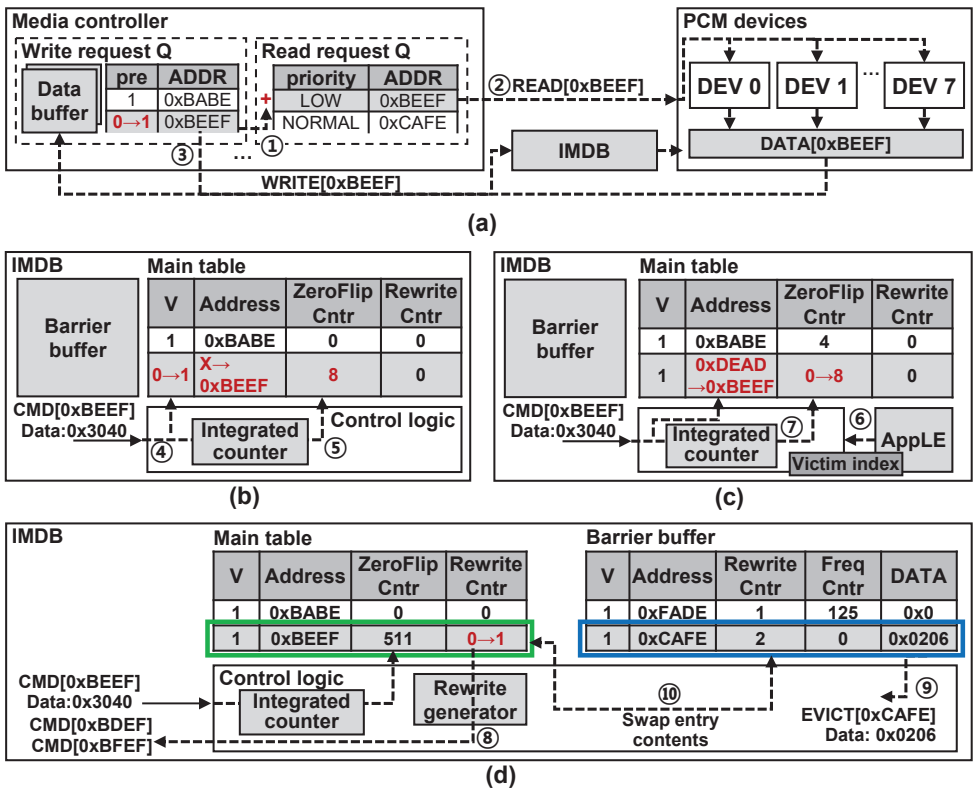


Figure 3.10: Operation of IMDB in different cases. (a) data preparation on the media controller, (b) Filling the non-empty main table, (c) replacement on the main table, (d) promotion and demotion.

In this section, a toy example is explained, combining all modules explained above. 0xBEEF (row=0xBE, column= 0xEF) is an assumed target to incur WDEs.

Data preparation. As shown in Figure 3.10 (a), the controller receives a write command. The command waits for the data already stored in the PCM device to prepare to count 1-to-0 flips in IMDB (①). Once the old data arrives through the read phase of the media controller (②), the command, along with new and old data, is

issued to IMDB (③).

Filling the non-empty main table. In this case, misses occur in both the barrier buffer and the main table; however, the main table may have a vacant space for the input command. Therefore, an entry is directly allocated for an address (i.e., "cold-start miss"), as shown in Figure 3.10 (b) (④). The number of zeros in the write data is recorded in *ZeroFlipCntr* as prior knowledge to prevent early eviction. If subsequent commands access the same address, the number of 1-to-0 bit flips is counted and added to the *ZeroFlipCntr* (⑤).

Replacement on the main table. In comparison to the previous case, the main table is full in this case, activating the replacement policy for a new address, 0xBEEF. As shown in Figure 3.10 (c), the entry of 0xDEAD has the minimum of *ZeroFlipCntr*, which is determined by AppLE (⑥). Subsequently, 0xDEAD is replaced by 0xBEEF (⑦).

Entry promotion and demotion. As shown in Figure 3.10 (d), when repetitive writes on 0xBEEF cause *ZeroFlipCntr* to reach the threshold, rewrite commands on neighbors (i.e., 0xBDEF and 0xBFEF) are sent to the media controller (⑧). Meanwhile, the barrier buffer demotes an entry to the main table. The eviction command is generated from the demoted entry and sent to the media controller (⑨). After the demoting and promoting entries are read, they are interchangeably stored. In particular, the promoting entry additionally stores data to further prevent WDEs (⑩).

3.5 Evaluation

3.5.1 Configuration

Table 3.2: Simulation configurations

Simulator	Device	Description
gem5	Cores	Out-of-order, 4-core, 2 GHz
	L1 cache	I-cache: 2-way set associative, D-cache: 4-way set associative, each has a capacity of 64 KB.
	L2 cache	Shared last-level cache. 16-way set associative, 1 MB.
NVMain	Media controller	Separated write queue and read queue (64-entry), FR-FCFS.
	PCM	Read: 100 ns, RESET: 100 ns, SET: 150 ns Write disturbance limitation: 1K Size: 8 GB (2-rank, 2-bank/rank)

As shown in Table 3.2, the environment is built upon NVMain [88]. The energy per access on PCM and CAM-based SRAM is obtained from NVSim and CACTI, respectively (i.e., both configured with 22 nm technology) [108], [105]. It should be noted that CAM-based SRAM is configured as a fully associative cache. The processor is configured according to a mobile processor; hence, a relatively small L2 cache is considered as the LLC [109]. This can lead to tougher memory traffic on the PCM. Nonetheless, we select workloads with a wide range of misses per thousand instructions (MPKI) (see Table 3.3). Therefore, evaluating the proposed method under various

Table 3.3: Information of workloads

Workloads	Description	MPKI
SPEC::bzip2	General compression	11.98
SPEC::sjeng	Artificial intelligence (chess)	0.89
SPEC::h264ref	Video compression	1.65
SPEC::gromacs	Biochemistry	5.49
SPEC::gobmk	Artificial intelligence (go)	6.65
SPEC::namd	Biology	1.09
SPEC::omnetpp	Discrete event simulation program	6.99
SPEC::soplex	Linear programming optimization	21.31
pmix1	Queue, Hashmap, B-tree, Skiplist	10.24
pmix2	Queue, B-tree, RB-tree, Skiplist	11.10
pmix3	Hashmap, RB-tree, Queue, Skiplist	8.95
pmix4	RB-tree, Hashmap, B-tree, Skiplist	10.12

forms of traffic is possible. In addition, this study conducts trace-based simulations, which is done in previous studies [60], [61], to reduce the simulation time because more than 400 experiments for sensitivity analyses are required. Because the traces are extracted in the system-emulation mode in gem5, OS-related writes do not exist in the traces.

Table 3.3 shows the workloads and associated MPKI on the last-level cache. Typical workloads from SPEC CPU 2006 consisting of various MPKI are evaluated. Furthermore, synthesized persistent workloads (prefixed as “pmix”), which perform random insertions and deletions, simulate realistic in-memory database workloads because persistent workloads would generally be executed under an NVM-based main

memory system [16], [17], [18].

3.5.2 Architectural Exploration

Design parameters, specifically the number of entries in the main table (N_{mt}), the number of entries in the barrier buffer (N_b), and the group size dedicated to AppLE (N_g), are crucial when seeking a cost-effective architecture for IMDB. As explained in the previous section, the latency of AppLE can be entirely hidden by the IDLE state of IMDB from N_g =32 (see Figure 3.7), which also holds for N_g <32. Moreover, 64 is determined as the maximum number of entries in the barrier buffer to guarantee that no more than 10% of the flush time (i.e., 100 us) is consumed. As a result, the trade-off function of IMDB is defined as follows:

$$T = W(N_{mt}, N_b, N_g) + A(N_{mt}, N_b) + S^{-1}(N_b), \quad where\ N_g \leq 32, N_b \leq 64 \tag{3.1}$$

where W, A, and S are the number of WDEs, the area, and the speedup (i.e., execution time normalized to the baseline [58]), respectively. Based on Eq (3.1), this section evaluates the effectiveness of the prior knowledge and determines the main table size (N_{mt}). Subsequently, sensitivity analyses concerning the number of entries in the barrier buffer (N_b) and the group size for AppLE (N_g) are conducted to determine the cost-effective parameters. Finally, these parameters are applied and compared to previous studies.

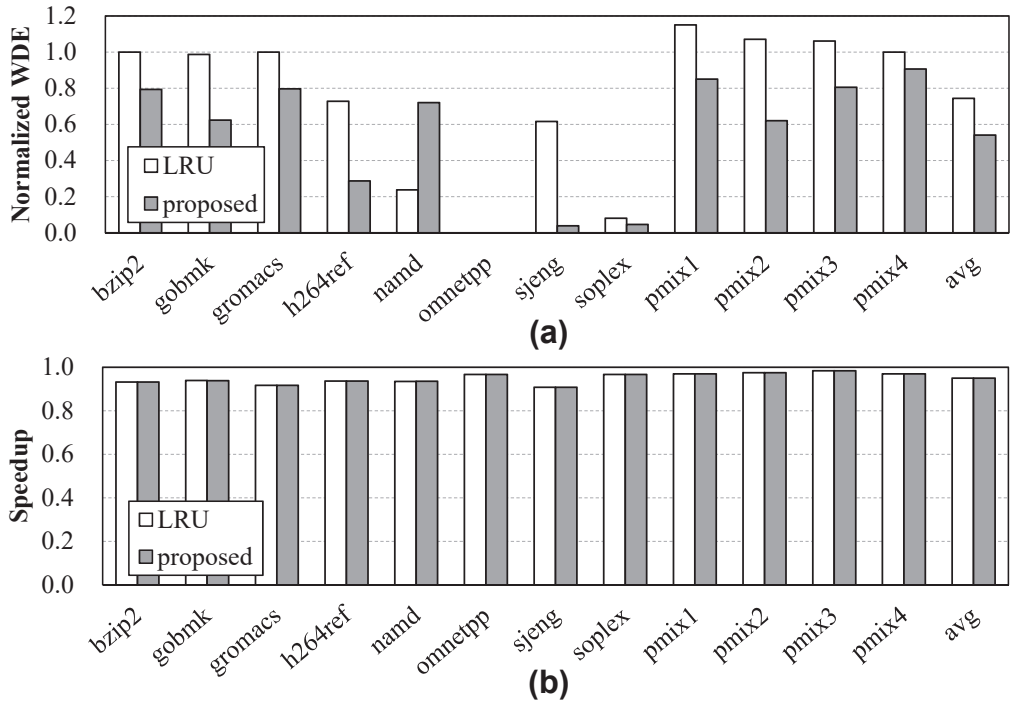


Figure 3.11: Performance according to different replacement polices. (a) normalized WDE, (b) speedup.

3.5.3 Effectiveness of the Replacement Policy

Before exploring the proposed architecture, verifying the effectiveness of the proposed policy is necessary. Figure 3.11 (a) shows the normalized WDE with different replacement policies (i.e., LRU and the proposed method). Please note that barrier buffers are not applied for a more straightforward comparison. The conventional LRU policy mainly retains recently accessed entries. Thus, the address close to WDEs can be evicted if it is not accessed for a long time. Many applications, such as *bzip2, gobmk, gromacs,* and persistent workloads, have this kind of access pattern; hence, the number of WDEs increases higher than the baseline. In contrast, the proposed policy observes the number of bit flips within *ZeroFlipCntrs* and evicts addresses having a few bit

flips. Furthermore, *RewriteCntrs* keep tracks of the long-term history by recording the number of restorations performed on WDE aggressors. Therefore, the proposed policy considers both short and long-term information, yielding lower WDEs than the LRU, as shown in Figure 3.11 (a). However, the LRU only shows 3× lower WDEs than the proposed policy on *namd*. This is because *namd* has high spatial locality and temporal locality. We find that *namd* achieves a 70% higher row buffer hit rate than an application of a similar MPKI (i.e., *sjeng*). Accordingly, *namd* achieves 7.7× higher hit ratio in the main table when the LRU is adopted. Therefore, the primary reason for the lower mitigation performance than the LRU on *namd* is the lower number of hits in the main table. However, such a performance gap on *namd* would be mitigated with parameter optimization in the following subsections.

Figure 3.11 (b) presents the speedup, which is defined as the execution time of the baseline over the execution time of the objective method [58]. Both methods provide similar performance outcomes, where the proposed method shows 0.002% lower performance than the LRU policy. From this perspective, the proposed replacement policy efficiently rewrites data vulnerable to WDEs and thereby yields far fewer WDEs without compromising the speed.

3.5.4 Sensitivity to Main Table Configuration

Figures 3.12 (a) and (b) show the normalized WDE regarding different numbers of entries in the main table. Both figures show that WDEs generally decrease as the number of entries increases. In particular, as shown in Figure 3.12 (a), while the number of WDEs exceeds that in the baseline when the number of entries is fewer than 256, the number decreases sharply from 2048 entries. This is because the small-sized table hardly manages data patterns and incurs unnecessary rewrite commands. On the

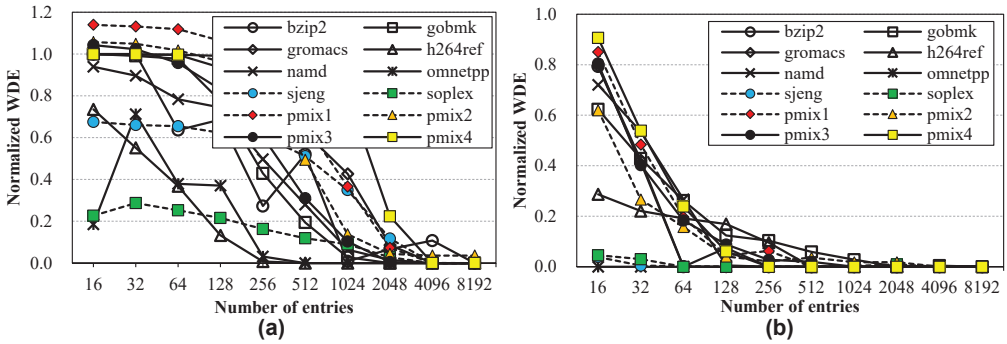


Figure 3.12: Normalized WDEs regarding different numbers of entries in the main table. (a) normalized WDEs without prior knowledge, (b) normalized WDEs with prior knowledge.

other hand, as shown in Figure 3.12 (b), the 256-entry main table with prior knowledge yields a result equivalent to that of the 2048-entry table without prior knowledge. In other words, the proposed method yields an eightfold increase in the efficiency of the WDE mitigation performance.

Figure 3.13 presents the average normalized WDE and the capacity required for the main table, and the probabilistic insertion scheme discussed in Section 3.2.2 is already adopted for both configurations. As shown in Figure 3.13, the normalized WDE is **95%** lower than the case without prior knowledge at 256 entries. Furthermore, the main table's capacity significantly increases from 512 entries; hence, 256 entries can be selected as an appropriate number of entries in the main table, considering the trade-off between the performance and the area. In summary, from this subsection, the number of entries in the main table is fixed at N_{mt} =**256**.

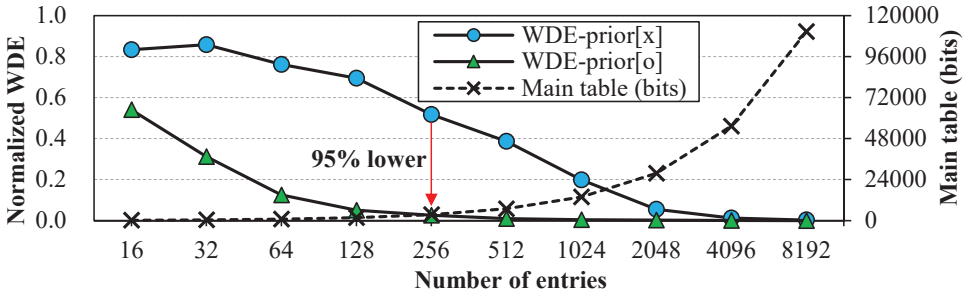


Figure 3.13: Average normalized WDE and SRAM capacity.

3.5.5 Sensitivity to Barrier Buffer Size

Figure 3.14 (a) shows the number of WDEs with different numbers of entries (i.e., size) in the barrier buffer. For clarity, the results are normalized to the *temporal base condition*; that is, the main table consists of 256 entries with the prior knowledge. Please note that Figure 3.14 (a) only shows benchmarks still having WDEs under the temporal base condition. As shown in this figure, most benchmarks yield significantly fewer WDEs with the 4-entry barrier buffer. On the other hand, WDEs in *gobmk* decrease when the 64-entry barrier buffer is applied because some addresses have extremely long-period write patterns, which are hardly concerned by the proposed policy regardless of the size of the barrier buffer. However, AppLE resolves this problem, as shown in the following subsection.

Figure 3.14 (b) shows the average normalized WDE of the benchmarks mentioned above. Because the speedup does not increase markedly according to the number of entries, S^{-1} is referred to as a constant in Eq (3.1). Furthermore, the capacity of the barrier buffer is at least three times as small as the main table for $N_b \leq 16$ (see bit widths of tables in Figure 3.4(a)), which makes the capacity of the barrier buffer negligible compared to the main table. Therefore, analyzing W in Eq (3.1) is sufficient to obtain a cost-effective architecture, thereby selecting $N_b = \mathbf{8}$ as the trade-off point

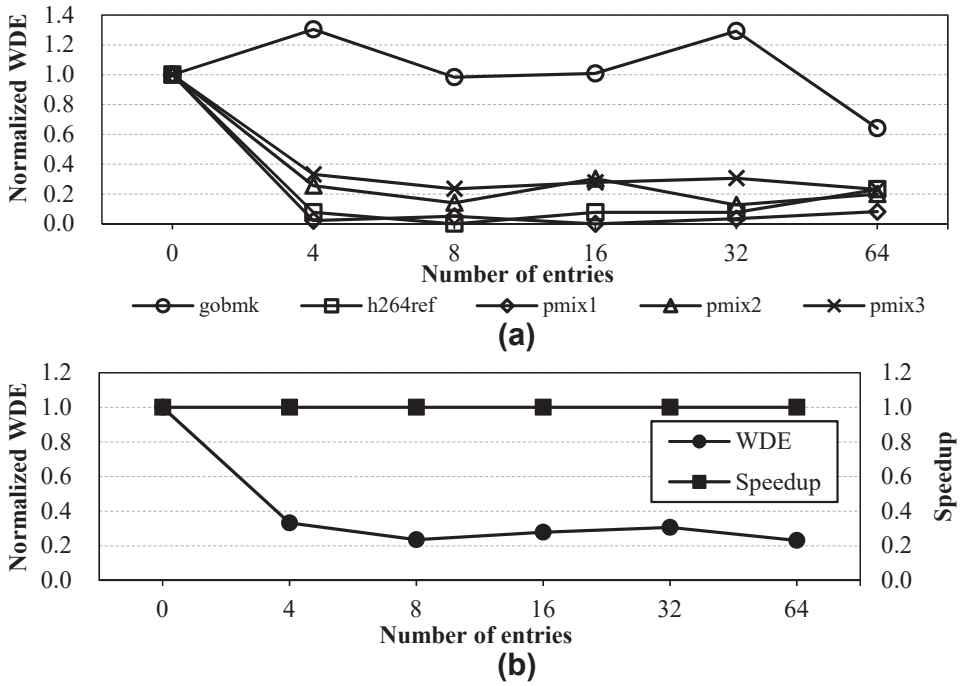


Figure 3.14: Sensitivity to the number of entries in barrier buffer. (a) normalized WDE, (b) average performance.

because the WDE decreases stably from 8 entries (i.e., **76.5%**).

3.5.6 Sensitivity to AppLE Group Size

Figure 3.15 (a) presents the absolute number of WDEs with different numbers of groups. Here, the barrier buffer is not applied for straightforward analysis, and 256 groups mean that AppLE is not applied. As presented in Figure 3.15 (a), WDEs lower with fewer groups for most benchmarks. Furthermore, AppLE has the potential for avoiding "tricky patterns". The worst-case behavior for WDEs can be caused by repetitive 0 and 1 pulses on the same address, which incurs WDEs on 512×2=1024 bits. However, the main table can easily detect such a pattern because it manages the number of 1-to-0 bit flips and generates rewrite operations on vulnerable addresses. In

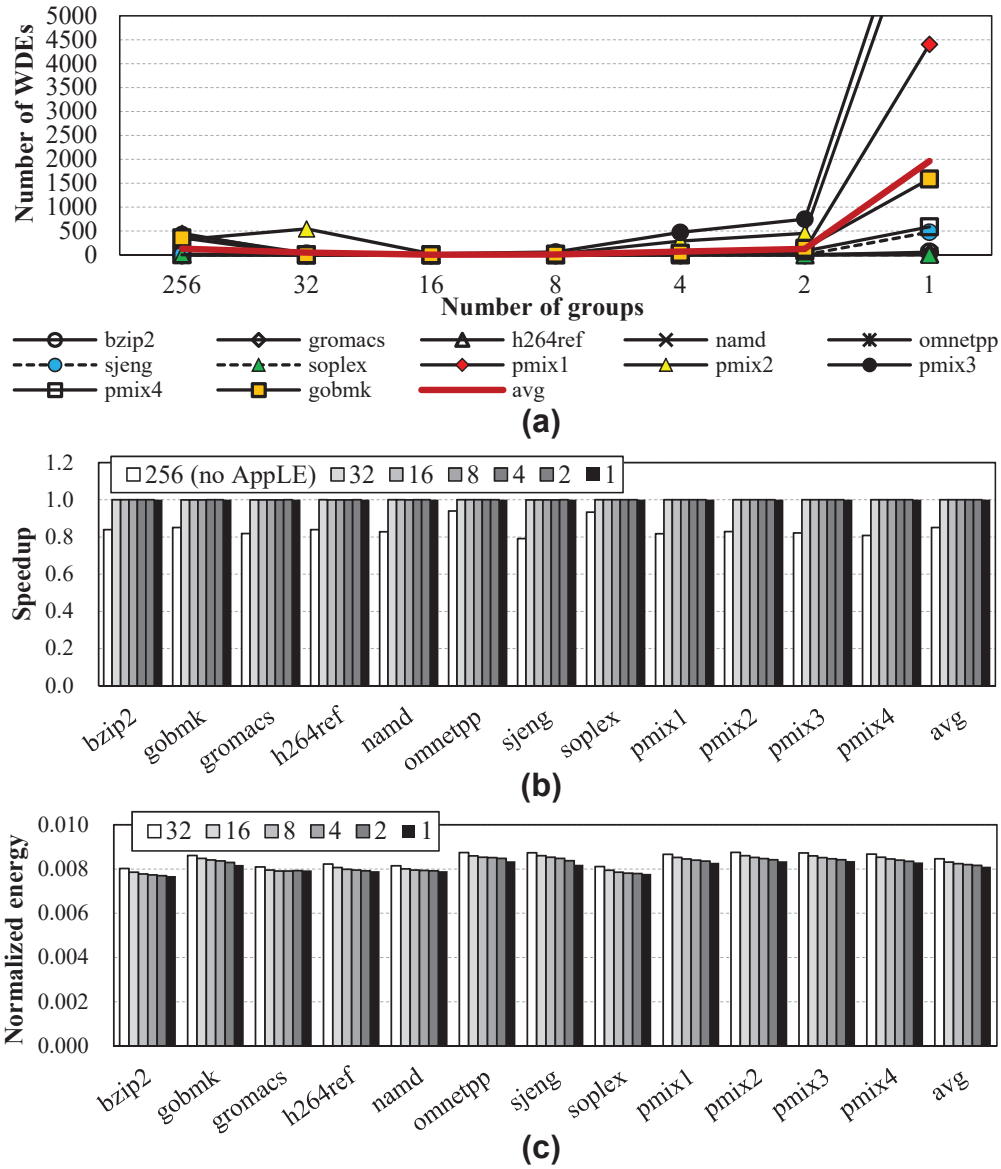


Figure 3.15: Sensitivity to group numbers in AppLE. (a) WDE, (b) speedup, (c) energy.

contrast, a trickier way to induce WDEs is incurring 1-to-0 bit flips on an address (say "A") with a long period (e.g., *gobmk*). Furthermore, a large number of unrepeated addresses except "A" are programmed in this long period (i.e., ABC...ADE...A...). This tricky pattern confuses the main table and frequently replaces entries; however, AppLE

binds a few entries as a group, and only one entry randomly becomes a replacement candidate within a group. Therefore, the adversarial address hardly gets evicted from the table for larger group size. The graph of *gobmk* in Figure 3.15 (a) shows that the group size of 8 (i.e., the number of groups is 32) yields lower WDEs than the case without AppLE. However, WDEs increase significantly from 2 groups on average (see bold red graph in the figure). In particular, the *fully randomized replacement policy* (i.e., one group) shows 15× more WDEs than the case without AppLE, indicating that the fully randomized replacement policy is less reliable than this study. As a result, we can select N_g =**4** or **8** as appropriate design parameters for AppLE.

Figure 3.15 (b) presents the speedup regarding different numbers of groups. In this study, AppLE is executed sequentially within the idle state. If AppLE is not applied, at least the latency of 256 cycles is induced by the comparison of counts. Even the latency can be hidden within the write latency (i.e., 120 cycles), at least 136 remaining cycles slow down the performance by 15%, as shown in the figure. In contrast, AppLE lowers the number of execution cycles, which is shorter than the write latency; hence, no performance degradation is incurred by the replacement policy.

Figure 3.15 (c) shows the energy consumption in the SRAM, which is normalized to the case without AppLE. In general, the energy decreases as the number of read ports shrinks. In particular, the energy decreases by 1.7% when the number of read ports lowers from 32 to 16. Furthermore, the SRAM energy accounts for 0.8% of the total energy. Thus, applying AppLE has no negative effect on energy consumption.

3.5.7 Comparison with Other Studies

From sensitivity analysis in previous subsections, the cost-effective IMDB is selected as IMDB(e256b8g8), which consists of 256 entries in the main table, 8 entries in the

Table 3.4: Performance of different mitigation schemes

Schemes	WDEs	Speedup	Energy
SIWC-size [62]	0.7276	1.0417	0.9467
ADAM [60]	0.5341	0.9807	1.1765
Lazy correction [58]	0.1925→0	0.3628	2.1773
SIWC-entry [62]	0.0885	1.0628	0.8951
IMDB(e256b8g4)	2.08E-3	0.9561	0.9937
IMDB(e256b8g8)	4.39E-4	0.9560	0.9941

barrier buffer, and a group size of 8. The group size of 4 is denoted as IMDB(e256b8g4). These configurations are compared to the following schemes (see details in Section 2.2):

- *Lazy correction* [58]: This scheme defers subsequent VnC by temporarily storing errors in an error correction pointer (ECP) chip. Each entry of the chip records multiple error pointers of one PCM line. In this study, we assume that 10 pointers are handled.

- *ADAM* [60]: This scheme aligns the compressed data in the device alternately to avoid data pattern that is vulnerable to WDEs.

- *SIWC* [62]: This scheme sparsely caches write data in an SRAM to reduce the number of WDEs. In particular, *SIWC-size* indicates that the SRAM capacity is identical to that of IMDB, and *SIWC-entry* holds entries in an amount equal to that of IMDB.

Write Disturbance Errors

The second column in Table 3.4 shows normalized WDEs. *SIWC-entry* presents 87.84% lower WDEs than *SIWC-size* (i.e., 0.0885 vs. 0.7276) because the mitigation performance strongly depends on the cache size. *ADAM* is effective only if the compression ratio exceeds 0.5; hence, *ADAM* shows inferior performance, 0.5341, on average. *Lazy correction* yields a normalized WDE value of 0.1925. However, it is noteworthy that *lazy correction* shows *temporal* WDEs in run-time, which can be corrected with ECPs.

In comparison to previous methods, IMDB(e256b8g4) reduces WDEs to 2.08E-3. Specifically, there are **256**× and **43**× fewer WDEs compared to *ADAM* and *SIWC-entry*, respectively. IMDB (e256b8g8) yields **1218**× and **202**× better WDE mitigation performance than *ADAM* and *SIWC-entry*, respectively. Moreover, these configurations show comparable WDE mitigation performance to the case that the main table consists of 2048 entries without barrier buffers. While a 2048-entry main table requires 108b×2048×4-bank =864KB of SRAM, the combinational approach yields superior WDE mitigation performance with 16KB of SRAM, which is four times smaller than *SIWC* (see Section 3.6).

Speedup

The third column in Table 3.4 presents the speedup compared to the baseline. *Lazy correction* shows the lowest speedup (i.e., 0.36×) due to at least four read operations even the performance is already enhanced with a high-cost ECP device. Although the proposed method rewrites two neighbors, these operations are performed in an on-demand fashion instead of incurring four read operations per write operation, as VnC does. Therefore, the proposed method can outperform *lazy correction*. The speed of *ADAM* degrades by about 2% due to encoding and decoding processes of FPC.

In *SIWC-entry* and *SIWC-size*, slightly higher performance is achieved because the SRAM serves commands.

On the other hand, two configurations of the proposed method degrade approximately 4% speed degradation on average due to pre-write read and rewrite operations, where the performance difference between the two configurations is only 0.3%. However, the waiting cycles for memory systems constitute 12% of execution time in the baseline, according to our evaluation. Consequently, the proposed method degrades the performance of the overall system only by **0.48%**. Although *SIWC-entry* shows a slightly higher speedup, the WDE mitigation performance is much worse than the proposed method, resulting in a system with low reliability. As a result, we can obtain a more reliable PCM-based computing system with negligible performance overhead.

Energy

The fourth column in Table 3.4 shows the normalized energy. *Lazy correction* consumes 2.18× more energy compared to the baseline due to overheads of VnC. It is 45.87% higher than *ADAM*, which is 1.18× higher than the baseline. Despite the full elimination of WDEs in *lazy correction*, *lazy correction* performs more operations than others, thereby consuming more energy. Meanwhile, the write cache of *SIWC-size* absorbs write operations on highly accessed addresses because persistent workloads have relatively high locality due to cache line flush instructions, reducing the write energy consumption. Thus, *SIWC-size* consumes about 5% less energy compared to the baseline. Furthermore, the energy can be lowered by about 10.5% compared to the baseline with a larger number of entries, as declared by *SIWC-entry*; however, it should be noted that the WDE mitigation performance is not as excellent as the proposed methods. Although IMDB (e256b8g8) presents 9% higher energy consumption com-

pared to *SIWC-entry*, this outcome is still **0.59%** lower than the baseline owing to the on-demand rewrite operation and the "tiny" barrier buffer. Furthermore, the proposed method consumes **54.4%** less energy than *lazy correction*. These outcomes demonstrate that the proposed method mitigates WDEs in a more energy-efficient way.

3.6 Discussion

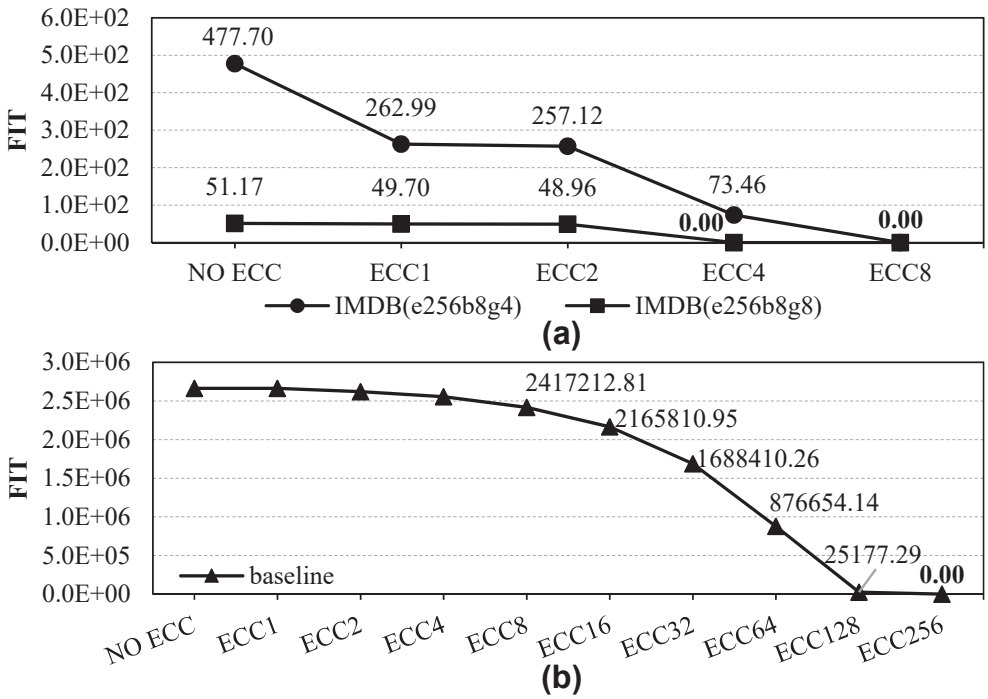


Figure 3.16: FITs when different ECC schemes are applied to (a) IMDB, (b) baseline.

Synergy with ECC schemes. In general, error-correcting codes (ECC) are proactively being employed in memory products that have reliability-related problems. In our case, ECC logic is placed on the media controller for system expandability. To observe the system reliability, we evaluated failure-in-time (FIT), which is the number of corrupted bits in an hour [110], [111]. Commonly, Figure 3.16 shows that FITs decrease when the correction capability of ECC enhances. In particular, Figure 3.16(a) shows that 0-FIT can be achieved when ECC4 (i.e., 4-bit error correction) and ECC8 (i.e., 8-bit error correction) are applied to IMDB (e256b8g8) and IMDB (e256b8g4), respectively. A (552, 512)-BCH code that is capable of correcting 4 errors [112] only incurs 1.5ns of latency (i.e., $<$1 cycle at 800MHz), according to the latency formula

in [113]. Therefore, only a minuscule amount of latency is required when IMDB is assisted by ECC. In contrast, evaluation results show that 0-FIT can be achieved only when the ECC capable of correcting 256-bit errors is applied to the *baseline*, as shown in Figure 3.16(b). However, such a correction capability is infeasible in practical products. In conclusion, IMDB can reduce the burden on the hardware area of ECC schemes significantly.

Discussion of SRAM capacity against SIWC. Considering the capacity of SRAM for the proposed method and a write cache-based study (i.e., *SIWC*) in a four-bank PCM system, the latter requires 256×64 B×4-bank=64 KB of SRAM if 256 addresses are managed per bank. On the other hand, for the proposed method, the main table entry has 25 b+8 b+72 b+3b=108b, and the barrier buffer entry has 64 B+25 b+8 b+8 b=553 b (see Figure 3.4. Therefore, the proposed method requires 256×108 b≈3.4 KB of SRAM on the main table per PCM bank, and the barrier buffer consumes 8×553 b≈0.6 KB of SRAM per PCM bank (see Section 3.5.5). Consequently, (3.4 KB+0.6 KB)×4-bank=16 KB of SRAM translates to 2 KB per 1 GB of PCM. If 256 addresses are managed, the proposed method consumes **4**× smaller SRAM area than *SIWC*, and the gap enlarges as the number of managed addresses grows. Besides the SRAM capacity, introducing SRAM as a data region requires considering the hold-up time constraint of supercapacitors. In particular, *SIWC* only holds dirty data; hence, flushing 256 volatile data requires 150 ns×256 flushes / 100us=38.4% of flush time at most (i.e., all row buffer miss commands on a single bank), where the value of 100us comes from [91]. In contrast, flushing data in the barrier buffer only requires 150 ns×8 flushes / 100 us=1.2%. In conclusion, the proposed design mitigates more WDEs without increasing energy or area from the supercapacitors.

Consideration of the security. Security problem should be considered for the ap-

plicability of the reliability scheme. A straightforward idea to attack the PCM with the proposed IMDB is exploiting or learning the replacement policy, which requires timing information of hit and miss on the main table [114], [115]. However, both types of latency are hidden within the write latency of PCM and appear to be equal from the user's perspective because AppLE can hide the miss latency as described in Section 3.3.2. Furthermore, the threshold value triggering rewrite operation must be known by the attacker to exploit the replacement policy. This is because we have to evict the objective entry before generating rewrite commands. However, the rewrite command is fed back to the media controller and rescheduled like a normal write command described in Section 3.2.2. In conclusion, triggering WDEs by exploiting the replacement policy in this "black box" is extremely difficult.

Handling power failure. In the proposed architecture, a strategy to handle power failures is required when using the barrier buffer. Because the main table only stores write patterns instead of data, data flush is not required. In contrast, the barrier buffer holds the overly flipped data and flushes contents upon system failure. When a power failure occurs, supercapacitors power the ADR domain [91], [103]. IMDB enters the flush mode and generates write requests with the barrier buffer. Requests are sequentially sent to the write queue in the media controller. Subsequently, all commands bypass the flush-mode IMDB to ensure data persistence. However, we have shown that 8 data entries in the barrier buffer are sufficient for the final design in Section 3.5.5, which is equivalent to only 1.2% of the time constraint of the supercapacitors. Therefore, the proposed method can be adopted with a low burden on the supercapacitors.

Expandability to other NVMs. The proposed method can easily be extended to other types of NVMs (e.g., ReRAM). In ReRAM, increasing the programming voltage will enhance the performance because the access latency is inversely proportional

to the voltage [116]. However, unselected cells are biased at half of the programming voltage on neighbors; hence, the resistances of such cells can easily be affected by higher biasing voltage. As a result, the resistance of a cell can be corrupted by a neighboring cell, like WDE in PCM. According to the earlier work [117], the voltage across the cell strongly depends on the position of the write drivers in the circuit; hence, many studies overcome write reliability issues in ReRAM by re-architecting the device [116], [118], [117]. Furthermore, [116] declares that this issue also results from accumulative write operations. Therefore, the proposed method can be applied to ReRAM as well if the rewrite threshold is moderately modulated.

3.7 Summary

This chapter proposes an on-demand table-based method reducing WDEs within a PCM module. The proposed method leverages SRAM tables to manage variations of write data, by which highly vulnerable addresses are rewritten. It declares that the table-based method requires a dedicated replacement policy, and prior knowledge of 0s in write data can enhance the WDE mitigation performance. Subsequently, AppLE efficiently downsizes the number of read ports on SRAMs that are incurred by the proposed policy to reduce both area and energy overhead incurred by the overloaded multi-port SRAMs. It is also demonstrated that the LRU and the fully randomized replacement policy are less reliable than the proposed method. Moreover, a tiny amount of SRAM absorbs further bit flips, allowing the offloading of the supercapacitor burden required on system failures. Consequently, several rigorous sensitivity analyses concerning design parameters are conducted to obtain a cost-effective architecture. The analysis shows that the proposed work reduces WDEs by 1218×, 439×, and 202× compared to *ADAM*, *lazy correction*, and *SIWC-entry*, respectively, while maintaining the speed and energy consumption that are similar to those of the baseline.

Chapter 4

INTEGRATION OF AN RMW MODULE IN A PCM-BASED SYSTEM

In the second dissertation, we propose an architecture that enhances the performance of the PCM-based system with an RMW module. PCM comprises memory cells that have a limited lifetime and higher access latency than DRAM. The page size of a PCM is preferred to be larger than 128 B 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 64 B, 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 chapter presents a new RMW architecture. The proposed model introduces a DRAM cache in the RMW module, which minimizes redundant read operations for

This chapter is an improved work that is originally published in IEEE Transactions on Computers, 2019 [90].

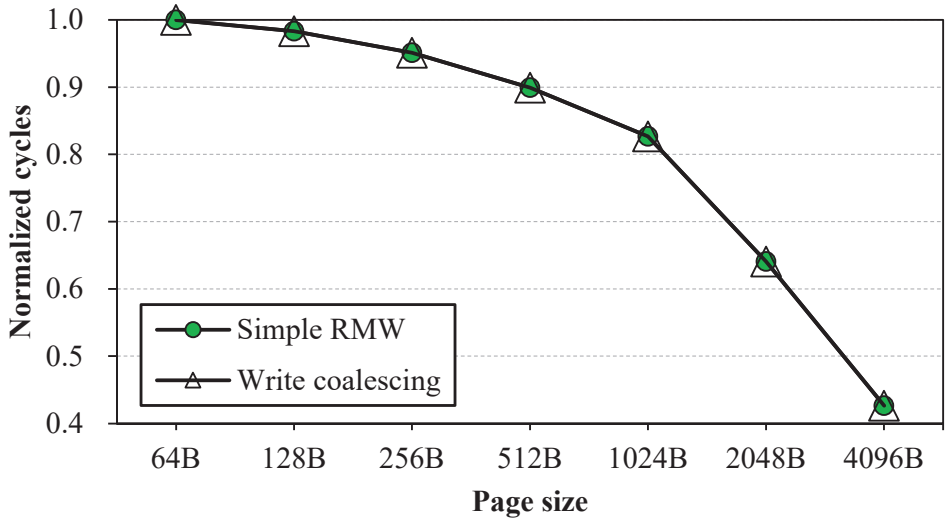


Figure 4.1: Normalized cycles when the RMW module is applied to the PCM-based system.

write operations by prefetching the entire transaction unit instead of merely caching the 64 B 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 4.2× and the read reliability by 58% as compared to the baseline.

4.1 Motivation

Effectiveness of the existing write coalescing scheme. Figure 4.1 shows the normalized cycles when the write coalescing scheme explained in Section 2.3.2 is applied to the PCM-based system, where general CPU applications (i.e., SPEC CPU 2006) are executed on the system. As shown in the figure, the write coalescing scheme yields only 0.01% speedup compared to the simple RMW explained in Section 2.3.1. We

found that the number of write coalescing accounts for 0.004% of total write commands. This is because the write coalescing scheme focuses on graphics applications, as explained in Section 2.3.2; however, normal CPU applications have few numbers of consecutive write commands. Furthermore, the commands of write-after-read and read-after-write are much more common in normal CPU applications. Thus, an RMW architecture in the PCM-based main memory that runs normal CPU applications must be devised to overcome the limitation of write coalescing schemes.

Unavoidable page size gap and its side effects. Page size in a PCM-based memory system must be larger than 64 B to enhance the throughput of the memory system, as explained in Section 1.2.2. In contrast, the cache line size of a general-purpose processor is 64 B, leading to speed degradation due to the page size gap. It is noteworthy that experiments held in previous PCM-related studies assume that the cache line size, commonly 64 B for a general-purpose processor, matches the page size of the PCM device without considering the non-symmetric case for practical usage [34], [10], [119], [120]. Furthermore, an RMW module in the PCM-based system substantially lowers the reliability due to read disturbance introduced by redundant read operations [65]. Thus, it is crucial to drill down a PCM-based system equipped with an RMW module.

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.

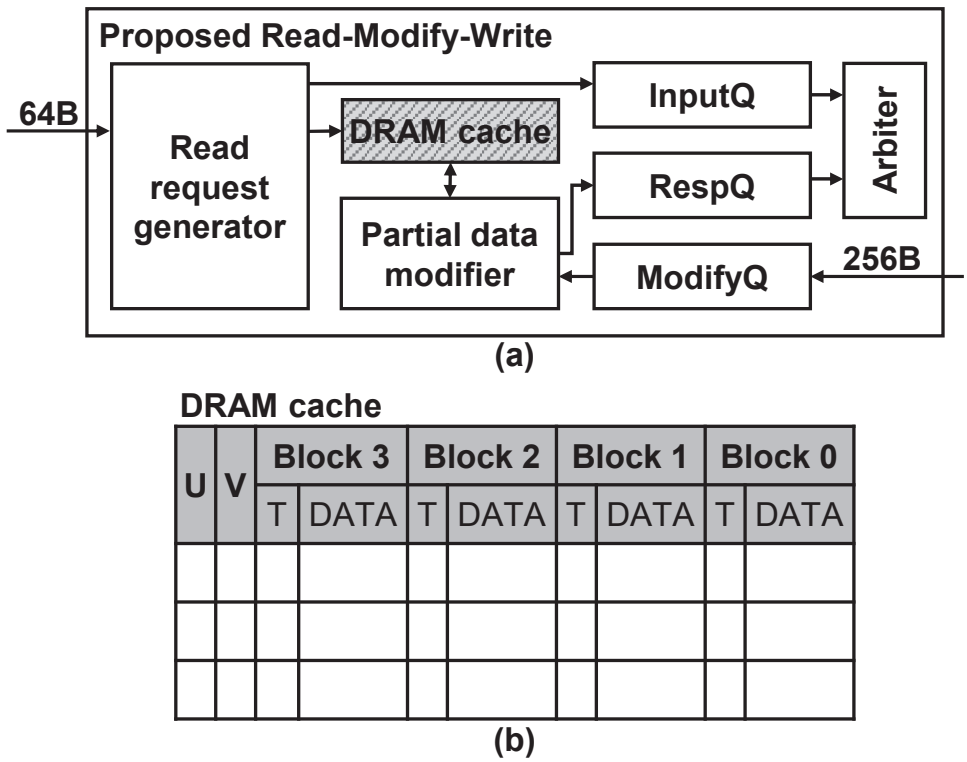


Figure 4.2: Structure of the proposed cache and its position in the RMW module.

4.2 Utilization of DRAM Cache for RMW

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 chapter.

4.2.1 Architectural Design

Figure 4.2 (a) shows an example organization of the proposed cache-based RMW. A 256 B page 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

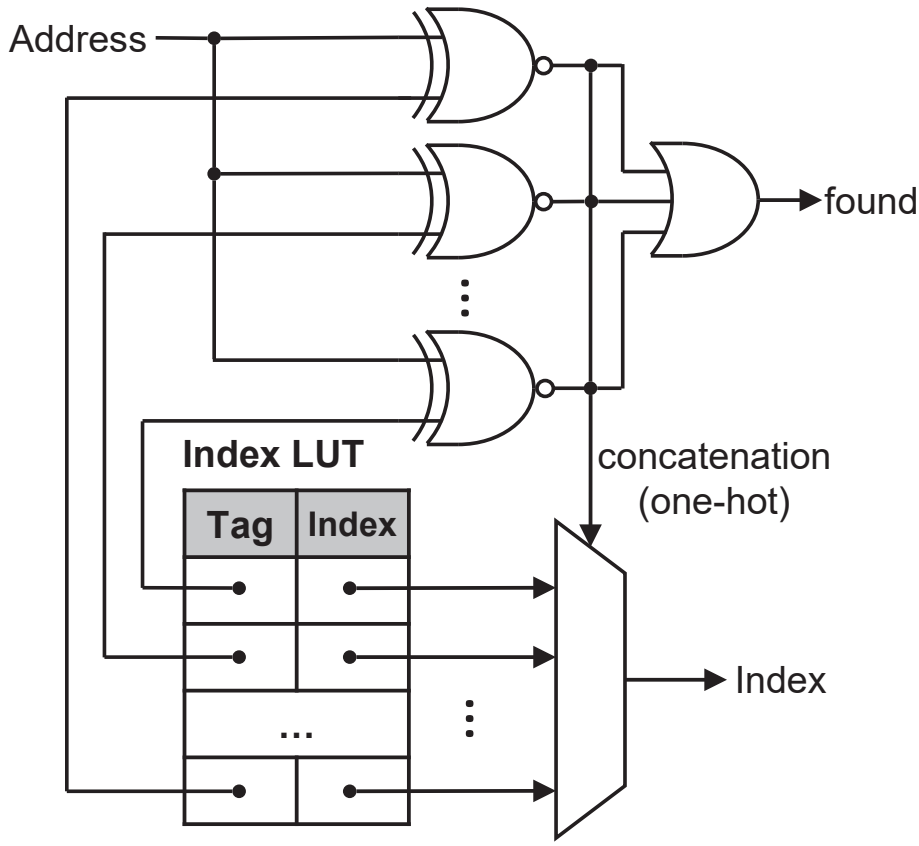


Figure 4.3: Address decoder of the cache in RMW.

data blocks, and *DATA* is a temporal field storing valid data. Therefore, the proposed cache does not need the data buffer in Figure 2.8. 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 page 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 Figure 4.2 (b) makes it possi-

ble 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 Figure 4.3. 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 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 [121] is organized as follows:

- RMW in [121]: the system defines DRAM as the 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 the main memory and additionally uses a private DRAM cache for the PCM, which is originally allocated for an address indirection table (AIT) that remaps the logical address to the

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
```

Figure 4.4: Pseudo code of the proposed cache-based RMW.

physical address. 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 resource overhead for the data buffer reduces with the existing resource.

4.2.2 Algorithm

A pseudo-code is presented to show the command process of the proposed structure as described in Figure 4.4. 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. 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 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 a large miss penalty (called a “stuck-in-queue” problem). In particular,

if the locality of an application is low, frequent cache misses result in performance degradation. A novel operation, called *typless merge*, to effectively mitigate this problem would be explained in the next section.

4.3 Typeless Command Merging

4.3.1 Architectural Design

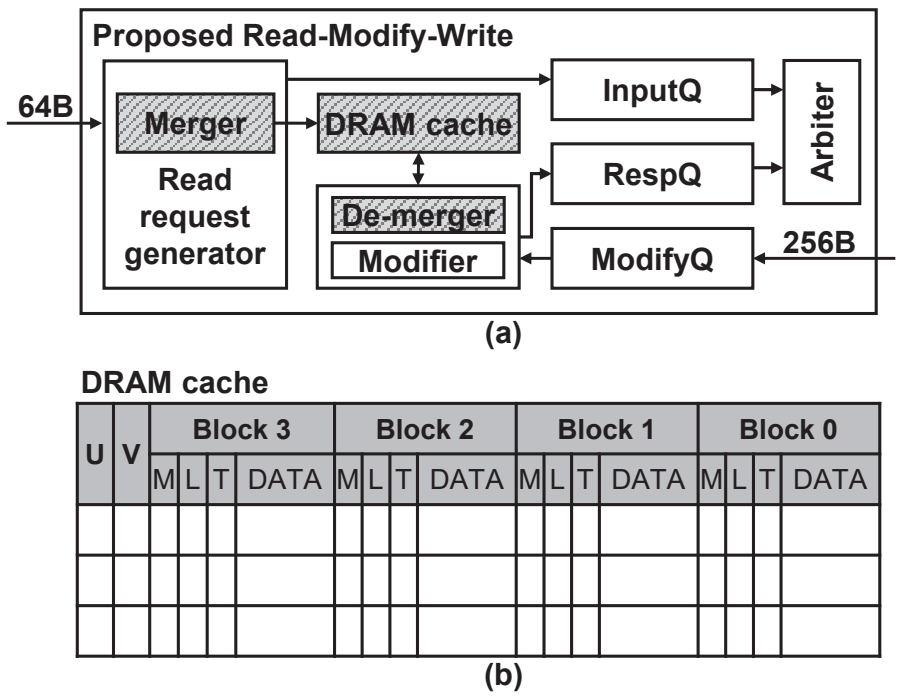


Figure 4.5: Cache table and RMW module supporting merge operation. (a) overall architecture, (b) modified data structure of DRAM cache region.

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*. Figure 4.5 (a) shows the overview of the modified architecture. Apart from the DRAM cache region that already exists in cache-based RMW, two more modules, namely *Merge* and *De-merger*, are added in the enhanced architecture. The former merges multiple commands regardless of command types and record the merging information into the DRAM cache. In contrast, the latter disassembles the

merged command for responses to the host.

Figure 4.5 (b) shows the details of the new entry structure and the modified RMW along with an example, which will be explained in detail in the next subsection 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*.

4.3.2 Algorithm

To implement the merge operation, the pseudo-code of Algorithm 1 in Figure 4.4 is slightly modified, as shown in Figure 4.6. Furthermore, pseudo-codes implementing both merge operation and de-merge operation (i.e., Algorithm 2 and Algorithm 3, respectively) are explained in Figure 4.7.

Modification of Algorithm 1: The operation handling the false case of the V-bit in Algorithm 1 is slightly modified (see Figure 4.6). 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.

Merger (Algorithm 2): As shown in Figure 4.7 (a), 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

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
```

Figure 4.6: Algorithm 1 for merge operation in which the modified part is shaded.

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 through-

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 and 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
```

(a)

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
```

(b)

Figure 4.7: Pseudo-codes for implementing (a) *Merger* and (b) *De-merger*.

put of the system can be significantly improved. As a result, the *Merger* can hide the miss penalty at the expense of simple logic, 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 Figure 4.7 (b), 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.

4.4 An Alternative Implementation: SRC-RMW

The implementation explained in the previous section (or [90]) is a straightforward approach without considering the hardware cost and its practicality. In particular, the implementation in [90] further requires an address decoder logic having a multi-ported SRAM on which the number of ports is equal to the number of entries of the cache. Therefore, such an SRAM is infeasible in the industry, as mentioned in the previous chapter. To overcome this challenging point, this dissertation further proposes a split region cache for RMW (SRC-RMW), which only uses the DRAM cache without introducing an additional SRAM resource.

4.4.1 Implementation of SRC-RMW

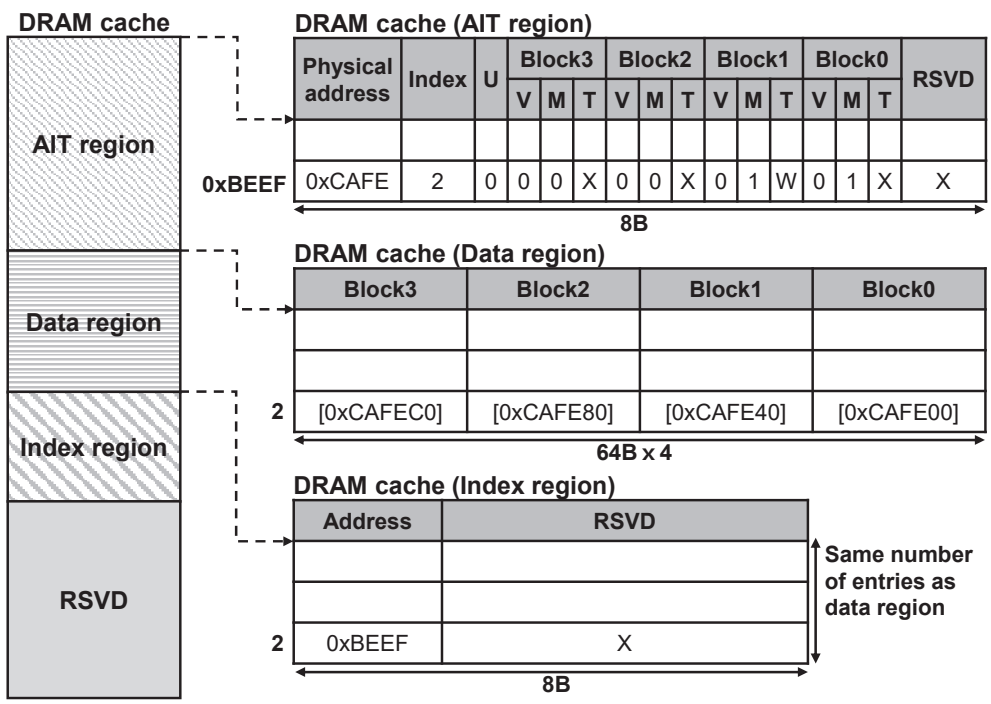


Figure 4.8: Implementation of SRC within a DRAM cache.

Figure 4.8 shows the implementation of SRC within a sole DRAM cache. SRC leverages two aspects of the DRAM cache that is originally dedicated to AIT. First, an AIT has 8 B for each entry in which only a few bits are used as the physical address field; hence, we can utilize the remaining reserved bits of an AIT entry to store the RMW metadata (e.g., M-bit mentioned in Section 4.3.1). Second, taking 8 GB of PCM as an example, the PCM controller requires 1 GB of AIT for translating each 64 B page. The size of AIT shrinks down to 256 MB if a PCM-based system adopts 256 B as a transaction unit. Therefore, we can leverage the remaining area (i.e., 768 MB) of the DRAM cache to store page data read from the PCM media. Consequently, SRC can be divided into three regions as follows:

- AIT region: It stores the physical address, index of the data region, and RMW metadata, including U-bit, V-bit, T-bit, and M-bit. Since RMW metadata is incorporated in the AIT region, both address translation and metadata reference can be performed simultaneously without accessing the DRAM cache twice. Please note that the metadata layout slightly differs from the layout in Figure 4.2 (b) or Figure 4.5 (b). V-bit is allocated for every block in a page because the page data and the metadata decouple in SRC. Therefore, it is also necessary to store the index information for directly accessing the data region without traverse.

- Data region: It merely stores page data that is brought from the PCM media.

- Index region: It stores the host address that is mapped to the index of the data region, which means this region has the same number of entries as the data region. This region is introduced for invalidating a data region entry with one DRAM access. For example, SRC-RMW needs to invalidate and replace the contents of index-2 that is assigned to the address-0xBEEF (physical address is 0xCAFE),

as shown in Figure 4.8. If no index region exists in SRC, the replacement policy must examine all index fields in the AIT region, leading to significant performance degradation. Therefore, the introduction of the index region can eliminate such a high-cost overhead by translating the index "inversely."

4.4.2 Design Constraint

The bit width of RMW metadata grows as both the capacity of PCM and the page size enlarge, thereby occupying more bits in an AIT entry. The total bit width, including the physical address and the RMW metadata, can be formulated as the following equation:

$$\begin{aligned} Bits_{AIT} &= Bits_{physicalAddress} + Bits_{Metadata}, \\ where\ Bits_{physicalAddress} &= \log_2 capacity - \log_2 pageSize, \\ Bits_{Metadata} &= \log_2 N + \frac{pageSize}{64} \times 3 + 1 \end{aligned} \tag{4.1}$$

where N denotes the number of data region entries. Moreover, 3 and 1 denote allocated bits for (V-bit, M-bit, T-bit)-pair per block data and U-bit, respectively. Figure 4.9 shows the bit occupation map in an AIT entry for an 8 GB PCM-based system, according to Eq 4.1. In this figure, the number of data region entries and the page size grow vertically and horizontally, respectively. The configurations in the red region are infeasible because the bit widths of these configurations are larger than 8 B (i.e., the full width of an AIT entry). Therefore, the page sizes for an 8 GB PCM-based system can be fixed as 128 B, 256 B, and 512 B, as shown in Figure 4.9 (a). Still, we have evaluated the performance of all possible page sizes as sensitivity analyses in Section 4.6. It is noteworthy that Eq 4.1 shows that the PCM capacity is wrapped by a logarithm operation, leading to less sensitivity by the capacity. Figure 4.9 (b) shows

8GB	64	128	256	512	1024	2048	4096
64	37	39	44	55	78	125	220
512	40	42	47	58	81	128	223
1024	41	43	48	59	82	129	224
2048	42	44	49	60	83	130	225
4096	43	45	50	61	84	131	226
8192	44	46	51	62	85	132	227
16384	45	47	52	63	86	133	228
32768	46	48	53	64	87	134	229

(a)

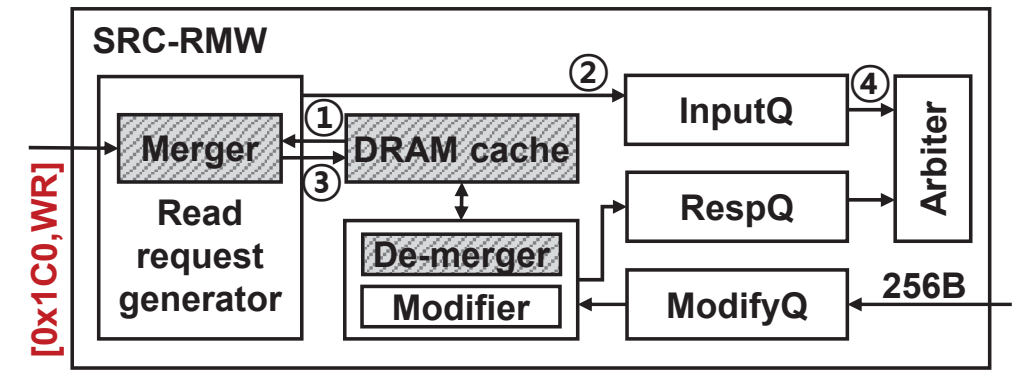
512GB	64	128	256	512	1024	2048	4096
64	43	45	50	61	84	131	226
512	46	48	53	64	87	134	229
1024	47	49	54	65	88	135	230
2048	48	50	55	66	89	136	231
4096	49	51	56	67	90	137	232
8192	50	52	57	68	91	138	233
16384	51	53	58	69	92	139	234
32768	52	54	59	70	93	140	235

(b)

Figure 4.9: Bit occupation map in an AIT entry for (a) an 8 GB PCM-based system, (b) a 512 GB PCM-based system.

the occupation map of a 512 GB PCM device, which is the largest capacity in currently available Optane products [86]. Our results in Section 4.6 show that 256 B is an appropriate page size considering the trade-off between speedup and energy consumption. Therefore, SRC-RMW is suitable for a wide range of PCM capacities.

4.5 Case Study



DRAM cache (AIT region)

	Physical address	Index	U	Block3			Block2			Block1			Block0		
				V	M	T	V	M	T	V	M	T	V	M	T
	③														
0x1	[0xCAFE]	0	1	1	1	W	1	1	W	0	0	-	0	0	-

Figure 4.10: A case study of merge process.

In this example, a PCM with a page size of 256 B 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, Figure 4.10 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

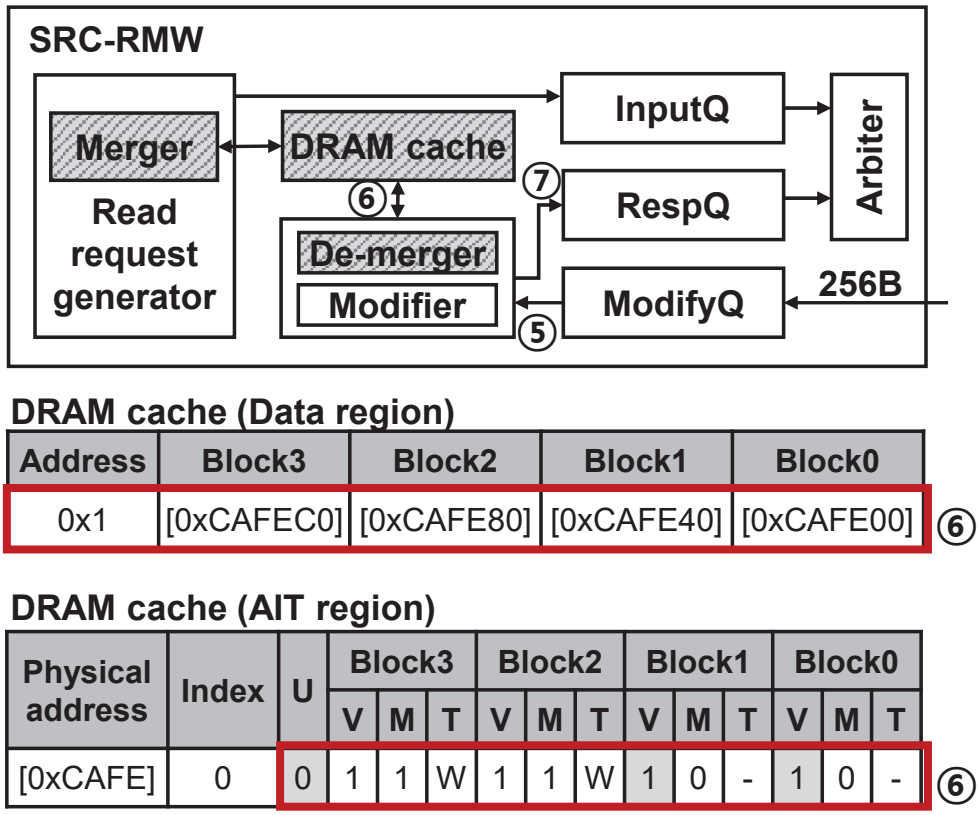


Figure 4.11: A case study of de-merge process.

of the incoming command and the cache entries.

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 predefined pending cycle exceeds, the 0x0-command is dispatched to the media controller.

When the response of the 0x0-command returns to RMW, it is first queued into the *ModifyQ* for the de-merging process. Figure 4.11 depicts the process of de-merging:

5. The 0x0-command at the head of the *ModifyQ* is redirected to the *De-Merger*

for command 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. Meanwhile, the U-bit and V-bit are set to 0 and 1 after decomposition, respectively.

7. Finally, the decomposed commands are stored into the *RespQ* for direct response or write-back to the host CPU or the media, respectively.

4.6 Evaluation

4.6.1 Configuration

Table 4.1: Simulation configurations

Simulator	Device	Description
gem5	Cores	Out-of-order, 4-core, 2 GHz
	L1 cache	I-cache: 2-way set associative, D-cache: 4-way set associative, each has a capacity of 64 KB.
	L2 cache	Shared last-level cache. 16-way set associative, 1 MB.
NVMain	Media controller	Separated write queue and read queue (64-entry), FR-FCFS.
	Interconnect	32-entry asynchronous FIFO
	PCM	Read: 50 ns, Write: 1000 ns Size: 8 GB (2-rank, 2-bank/rank)
	DRAM cache	Read: 10 ns, Write: 10 ns Replacement policy: LRU

In this chapter, two simulators are used to build the baseline system. First, an out-of-order 4-core processor with L2 last level cache (LLC) having 64 B line size is configured in *gem5* based on ARM Cortex A53 as shown in Table 4.1. Subsequently, the PCM system with memory controller, banks, and rank models is built with *NVMain* [88]. The RMW module is additionally implemented for cycle-accurate simulation of RMW operations. For the configuration of timing parameters in NVMain, the read latency (i.e., tRCD, or row-to-column delay) is set to 50 ns; the wite latency (i.e., tWP, or write pulse time) is set to 1000 ns. Please note that the **baseline** for normalization

Table 4.2: Information of synthesized workloads

Name	Included benchmarks
SPEC::mix1	*lbm, leslie3d, astar, gcc*
SPEC::mix2	*lbm, leslie3d, astar, bzip2*
SPEC::mix3	*leslie3d, astar, bzip2, gcc*
SPEC::mix4	*astar, bzip2, gcc, GemsFDTD*
SPEC::mix5	*mcf, lbm, gcc, bzip2*
SPEC::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*

is the model having a 64 B page size and a DRAM cache.

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

As shown in Table 4.2, 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 bench-

Table 4.3: 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

marks 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 [122], [40], [123]. In this study, applications with high MPKIs *lbm, leslie3d, mcf, gcc*, 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 4.1, 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 4.3, 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 [23]. These data structures are similar to those in the previous studies for persistent operations [23], [16], [17], and skip-list is a basic data structure of *ZSET* (or *sorted sets*)) in Redis [2]. 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 4.2.

4.6.2 Speedup

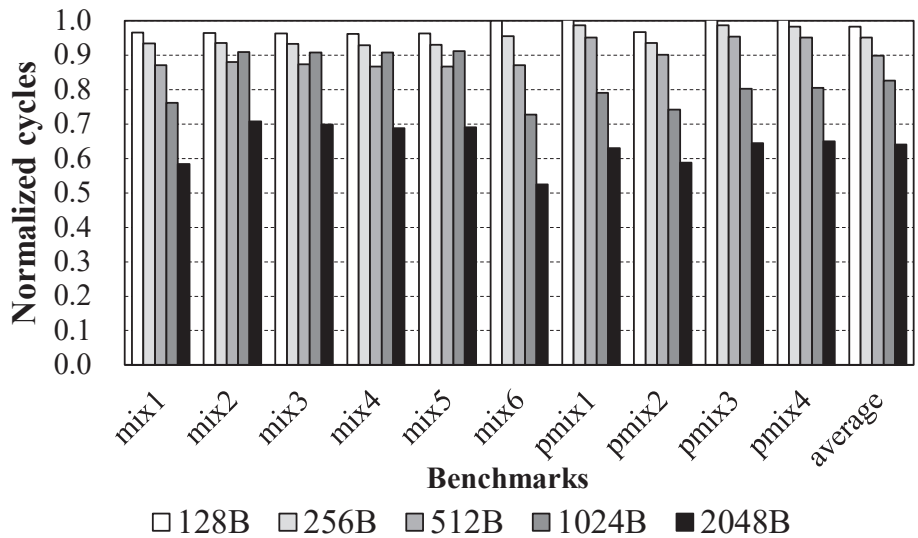


Figure 4.12: Speed degradation rate with respect to different page size where 64B is the baseline. The lower rate the worse performance.

The speed of the traditional RMW degrades as the page size increases because a memory transaction of a larger buffer size requires a longer burst length. The simulation results in Figure 4.12 verify the degradation. As shown in the figure, merely enlarging the page size to 2 KB and applying the traditional RMW (i.e., Section 2.2.1) degrades the performance by nearly 40%.

Figure 4.13 shows average speedup with respect to different numbers of entries in the data region when the page size ranges from 128 B to 2 KB. The figure shows that speedup saturates at 16K entries when the page size is 4 KB. On the other hand, 32 K-entry also becomes a saturation point when the page is smaller than 512 B. Since we have explained that the feasible page sizes of SRC-RMW are 128 B, 256 B, 512 B in Section 4.4.2, 32 K is selected as the final number of entries for SRC-RMW.

SRC-RMW achieves a speedup with an increase in the page size, as shown in

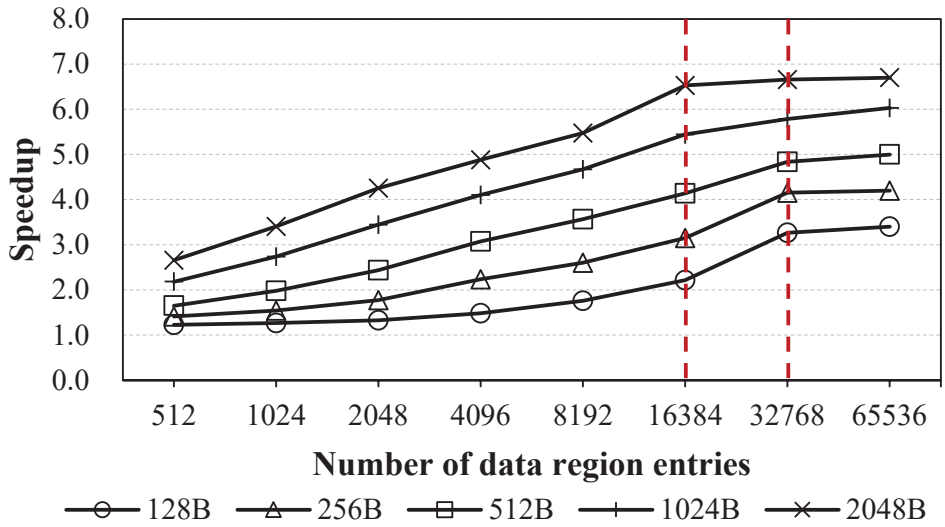


Figure 4.13: Average speedup concerning different numbers of data region entries.

Figure 4.14. In each graph in Figures 4.14 (a)-(d), the left sub-graph shows the result without typless merge, whereas the right sub-graph represents the improvements made by the merging operations. As shown in the left sub-figures of Figures 4.14 (a)-(d), the speedup increases as the cache capacity enlarges thanks to the enhanced cache hit rate. The average speedups of 256 B page systems are 1.7×, 1.9×, 2.3×, and 2.7× when there are 4 K, 8 K, 16 K, and 32 K entries in the DRAM cache, respectively. The maximum improvement is 5.2× compared to the baseline on average when the entry number is 4 K with 2 KB page size.

Some benchmarks have speedups smaller than 1 when the entry number is smaller than 1024 for 128 B page. This degradation occurs due to the ”stuck-in-queue” problem as discussed in Section 4.2.2. To mitigate “stuck-in-queue” problem, the typeless merge operation is applied to the SRC. The right sub-figures of Figures 4.14 (a)-(d) show the speedup of the typeless merge operation ranging from 128 B to 2 KB page size. As shown in these figures, the merge operation enhances the performance

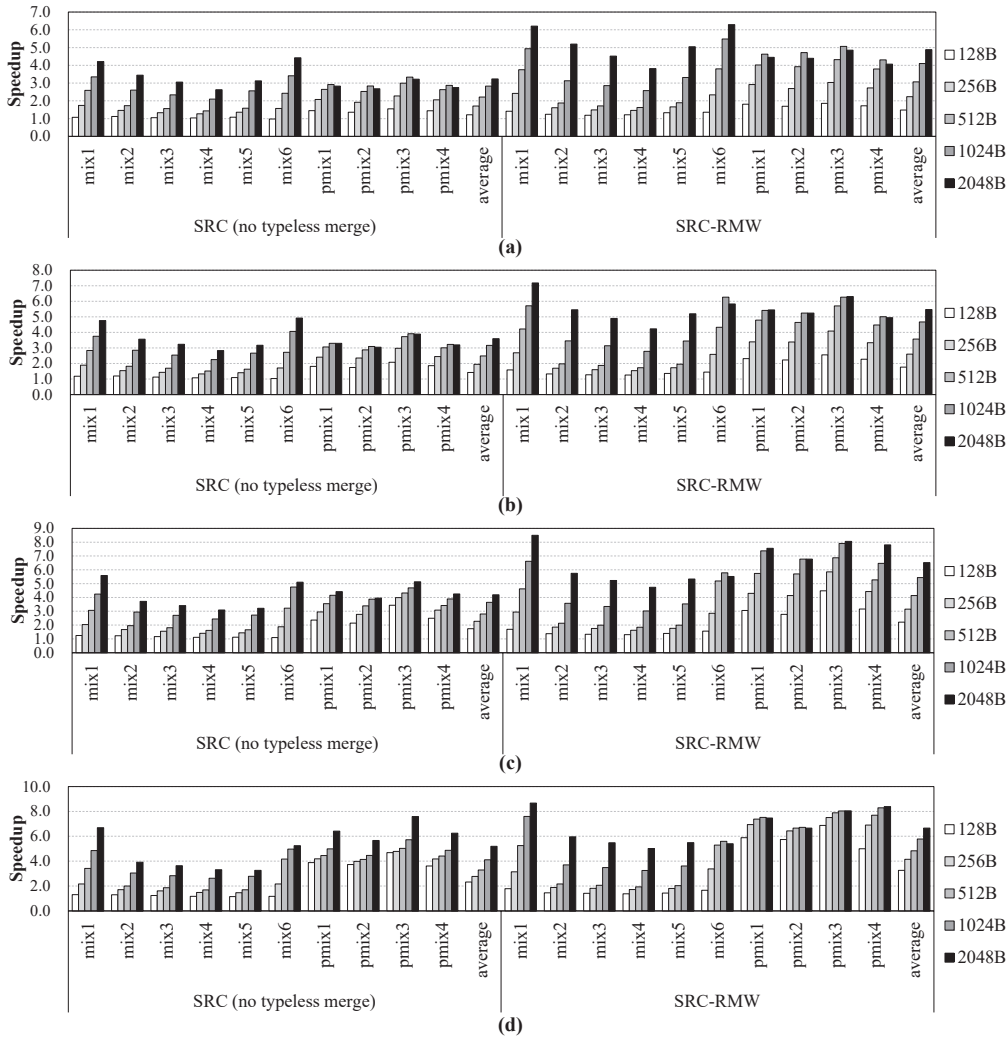


Figure 4.14: Speedup comparisons concerning different DRAM cache entry numbers. (a) 4K-entry, (b) 8K-entry, (c) 16K-entry, (d) 32K-entry.

markedly. For a system with 256 B page, the speedup achieves 2.2×, 2.6×, 3.2×, and 4.2× when the numbers of the cache entries are 4 K, 8 K, 16 K, and 32 K, respectively (see Figures 4.14(a)-(d)). The maximum speedup is 6.7× on average with 2 KB page and 32 K-entry cache, which is significantly higher than the value of 5.2× that is without typeless merge operation.

4.6.3 Read Reliability

The proposed system yields significant performance improvement. Still, it is still insufficient to show whether the proposed method is suitable for general use concerning reliability or not. Another unexpected benefit of the cache-based RMW is device reliability. Read disturbance error (RDE) is a self-disturbance error caused by a glitz on the sense amplifier. An RDE is fundamentally incurred by frequent and redundant read operations on a specific cell itself. Since a cache filters out most of the commands accessing neighboring addresses to prevent the commands from entering the PCM device, the number of RDEs can be highly reduced by applying the proposed method.

Figure 4.15 shows the average normalized RDEs that represent the ratio of the bit errors generated by SRC-RMW against the baseline. To cover technology variations, six different bit error rates are chosen, i.e., 1E-2, 1E-3, 1E-4, 1E-5, 1E-6, and 1E-7. The normalized values are represented by the bar graphs. The horizontal axis represents the page size, whereas four cache sizes are chosen as 4 K, 8 K, 16 K, and 32 K entries, respectively. The four graphs in Figure 4.15 represent the results for these four cache sizes, respectively. In general, the number of RDEs decreases with the increase of the DRAM cache entries. This is because more entries in SRC-RMW can provide a high hit ratio, thereby lowering the number of read operations on the PCM media. Thus, a small number of entries may yield higher RDEs than that of the baseline (e.g., 128 B page with 4 K DRAM cache entries). Furthermore, the number of RDEs also decreases with the increase of the page size. This is because the proposed model allows the controller to prefetch and reuse the data as the size of the page increases. In particular, a 256 B page shows 32% fewer RDEs than that of the 128 B page when the number of entries is 4 K with the bit error rate of 1E-5. For a 256 page tested at the bit error rate of 1E-5, the average errors are reduced by 30.2%, 39.9%, 49.4%, and 58.8% when

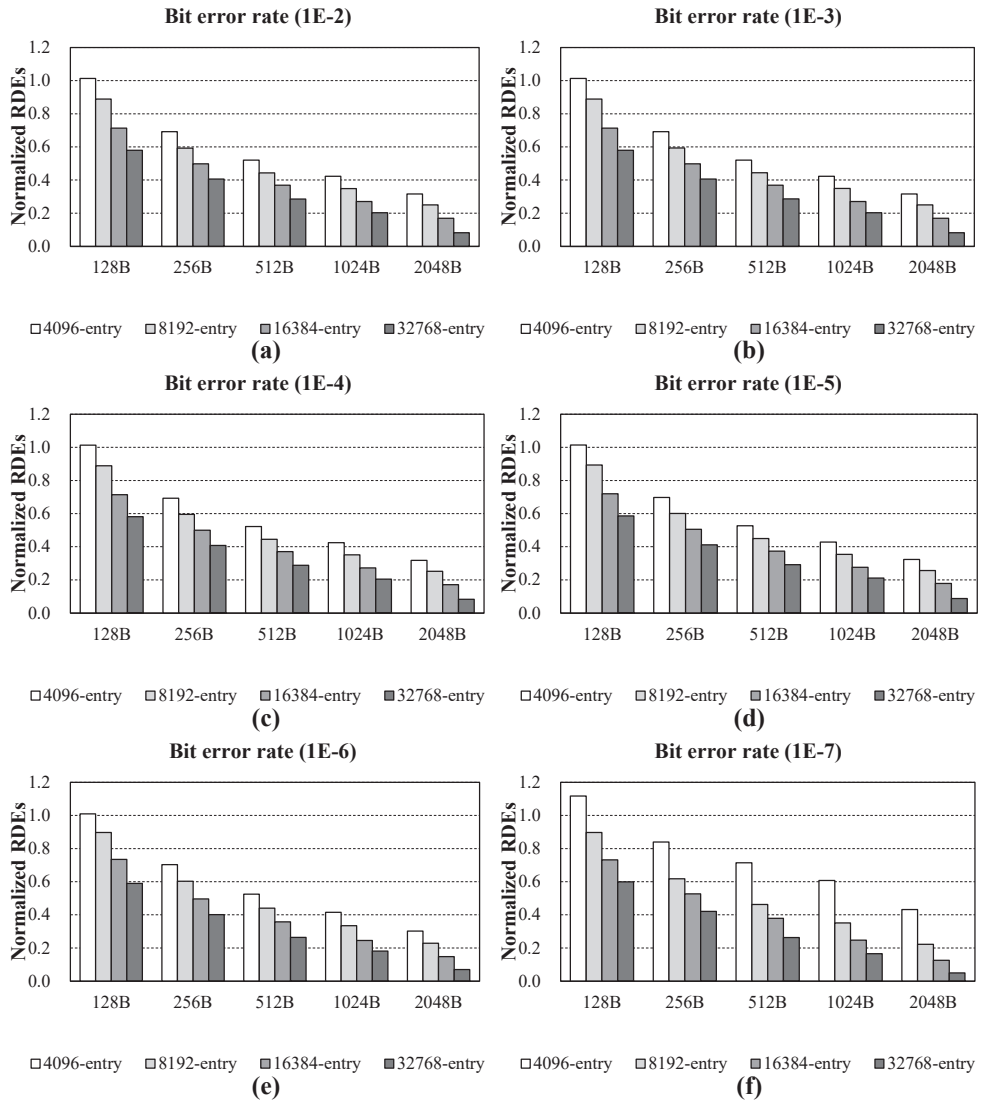


Figure 4.15: Average normalized bit errors and bit errors/read command after applying typeless merge concerning different DRAM cache entry numbers. (a) BER=1E-2, (b) BER=1E-3, (c) BER=1E-4, (d) BER=1E-5, (e) BER=1E-6, (f) BER=1E-7.

the cache has 4 K, 8 K, 16 K, and 32 K entries, respectively, as compared to that of the baseline. It is noteworthy that the variation of RDEs for different bit error rates is similar to that of 1E-5.

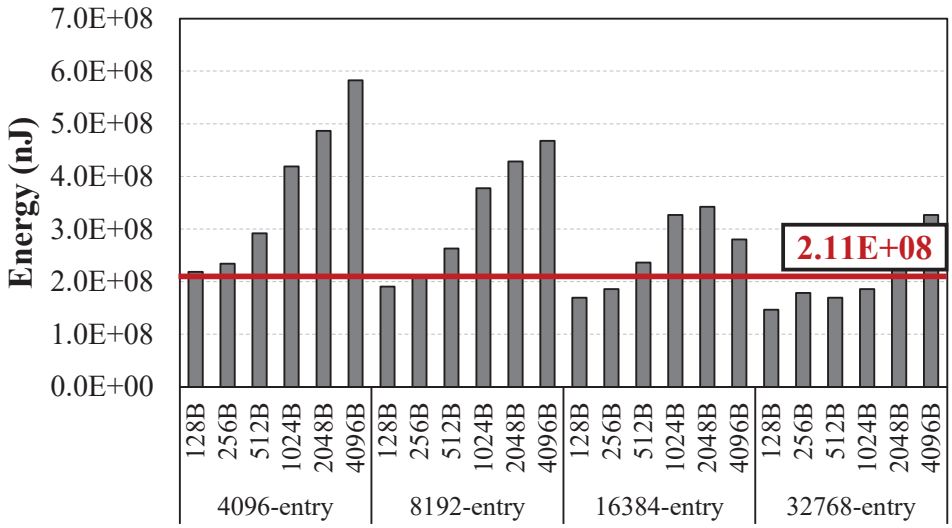


Figure 4.16: Average energy consumption of the proposed merge-operation where the red line shows the average energy consumption of the baseline in this chapter.

4.6.4 Energy Consumption: Selecting a Proper Page Size

Figure 4.16 shows that the average energy consumption of a memory system generally increases with the page size by adopting the SRC-RMW regarding different numbers of DRAM cache entries. Furthermore, the energy consumption of different page sizes is also 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 K-entry DRAM cache with 256 B page, the increase of the energy consumption of SRC-RMW is about 11% while the speed is improved by 2.2× compared to that of the baseline. The increase in energy consumption is relatively small because the main memory consumes about 11% of a whole computing system [124]. As a result, the proposed method can achieve 2.2× of the speedup with a relatively small increase in energy consumption (about 1.21%=11%×11%) compared to the baseline.

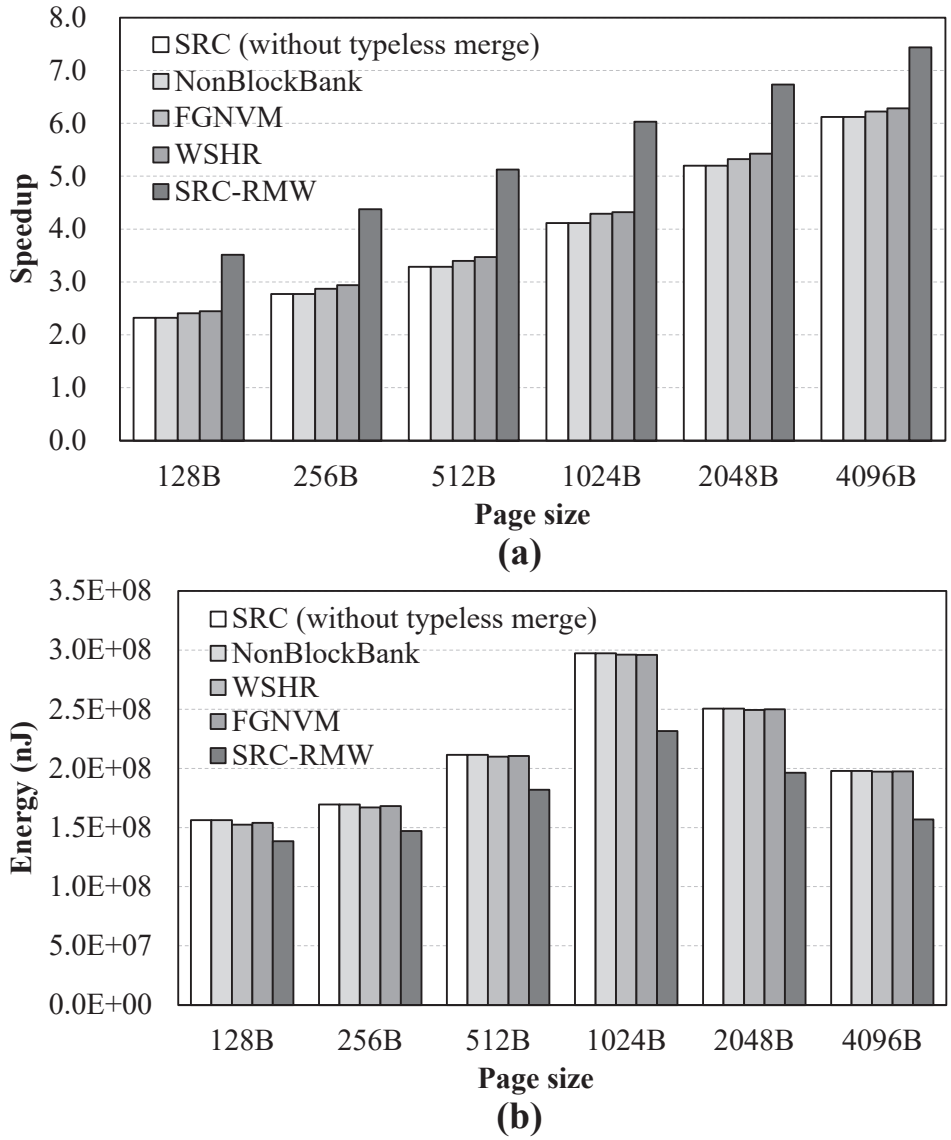


Figure 4.17: Comparison with different performance enhancement schemes. (a) speedup, (b) energy consumption.

The page size should be chosen to achieve the best trade-off between energy consumption and speedup. As shown in Figure 4.16, energy consumption increases for most entry options when the page size is 512 B. For example, 4 K-entry DRAM cache

with 512 B page yields 38.65% more energy consumption compared to the baseline, which is 27% higher than that of 256 B page. Since the number of entries can be reduced as the system configuration (e.g., the resource usage of DRAM cache) changes, it is necessary to choose a page size that yields notable speedup with a low energy overhead. Therefore, 256 B is chosen as the optimal page size.

4.6.5 Comparison with Other Studies

It is necessary to show how SRC-RMW enhances the performance of RMW compared to previous schemes. Therefore, this chapter applies the following schemes to SRC-RMW rather than the traditional RMW for fair comparison:

- NonBlockBank [77]: A non-blocking PCM bank (NonBlockBank) 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 a write-precedence scheduling policy.

- FGNVM [125]: It is a device design that enables multiple activations at tile-level, called Fine-Grained NVM (FGNVM), is proposed to exploit parallelism for hiding more latency.

- WSHR [51]: It shows that the latency of write operation can be hidden by adopting "reads-under-write" methodology. This method introduces a write-status hold register (WSHR) in a memory controller and four pages for executing multiple read commands in an overlapped manner (based on the LPDDR2 standard). It leverages the stabilization time of write operations on PCM devices to hide the latency by accessing multiple pages in a non-blocking way.

The average speedups and average energy consumption of all schemes, including proposed methods, are shown in Figure 4.17 (a) and (b), respectively. As shown in Figure 4.17 (a), WSHR shows the highest speedup among previous schemes; however, the proposed method (i.e., with typeless merge operation) achieves a 32.8% improvement compared to WSHR. For energy consumption in Figure 4.17 (b), all schemes show similar energy consumption except SRC-RMW without typeless merge operation. SRC-RMW consumes 11.9% lower energy than WSHR due to the decrease in the number of processing read commands. Therefore, our typeless merge operation yields the best performance concerning both speed and energy compared to previous schemes.

4.7 Discussion

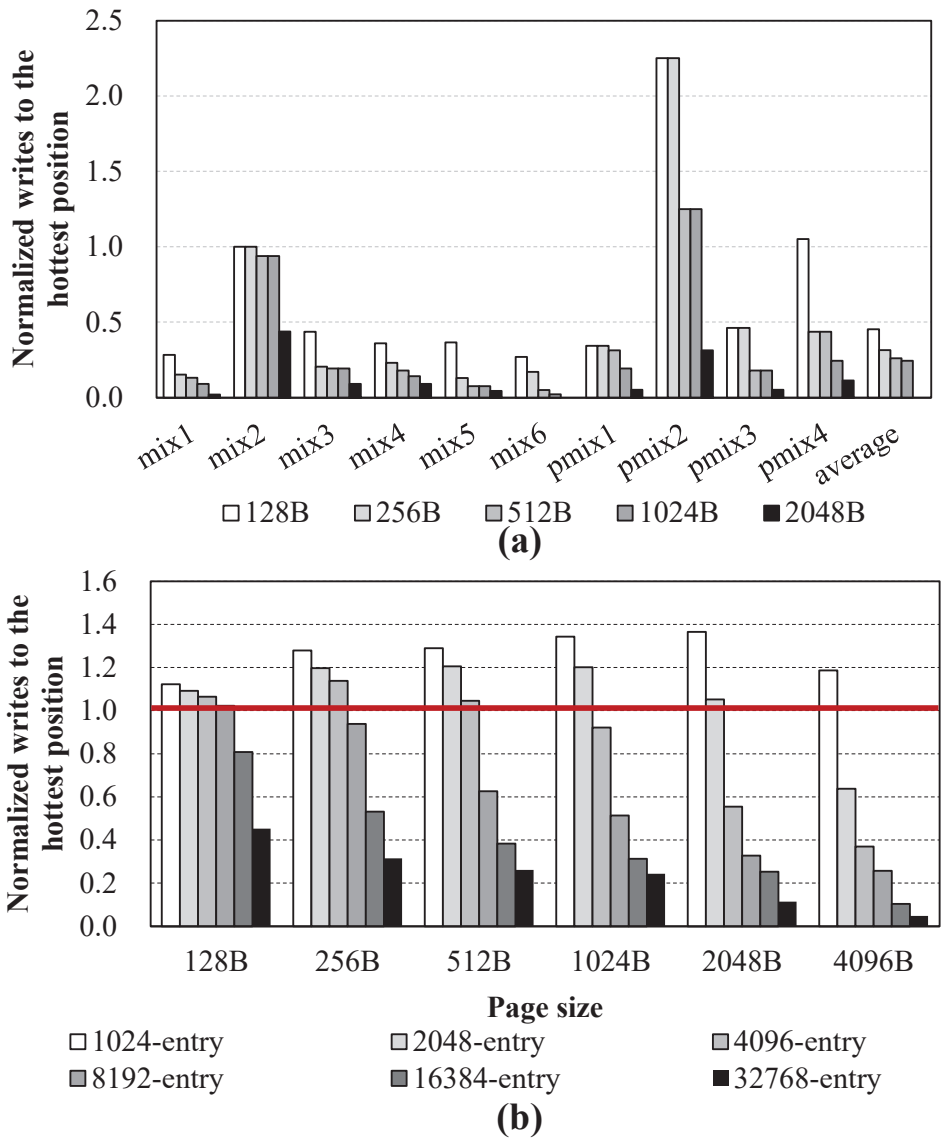


Figure 4.18: Normalized writes to the hottest position of the proposed merge-operation: (a) the results for each benchmark when DRAM cache has 4096 entries, (b) the results for different DRAM cache entries.

Effects on 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); this metric is used for quantifying the effectiveness of wear-leveling in [126]. Figure 4.18 (a) shows the writes to the hottest position of the proposed method equipped with a 32 K-entry DRAM cache which is normalized to the baseline by increasing the page size from 128 B to 2 KB. The wear-out reduces if the page size enlarges from 128 B to 512 B, owing to the increasing data prefetching effect on the DRAM cache. The proposed method with a 256 B page and 32 K-entry DRAM achieves 68.6% reduction of the worst-case wear counts compared to the baseline. Figure 4.18 (b) also shows that the proposed method prevents wear-out more effectively as the DRAM cache size increases. In particular, an 8 K-entry DRAM cache yields lower wear-out for all page sizes compared to the baseline; hence, 8 K-entry becomes the starting point that ensures higher system reliability than that of the baseline.

Relevance of the page size concerning the cost. The selection of the page size in Section 4.6.4 is compliant with the previous research that also chooses the transaction unit to be no more than 256 B [10], [34], [39]. The selected trade-off increases the speed by 2.2× and reduces total error occurrences by 58.8%, as it increases the energy consumption by 11% when compared to the baseline for the 32 K-entry DRAM cache with the typeless merge operation. The hardware cost of the merge operation is very small because the 32 K-entry DRAM cache demands about 264 MB (256 MB for AIT region + 8 MB for data region and index region), which saves 760 MB of DRAM cache compared to the baseline (i.e., 64 B-page system).

4.8 Summary

This chapter first proposes an RMW architecture utilizing a DRAM cache for enhancing read reliability and performance in a PCM-based system. However, it may induce an unexpected performance degradation for some workloads due to the high miss penalty of the cache in the PCM. Therefore, *typeless 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 256 B page system with a 32K-entry DRAM cache enhances the speed and read reliability by 2.2 $\times$ and 58.8%, respectively, on average. The DRAM cache requires approximately 8 MB of storage space, of which cost is very small, as discussed in the previous section.

Chapter 5

AN ALL-INCLUSIVE SIMULATOR FOR A PCM CONTROLLER

In the last dissertation, we propose an architecture simulator that simulates a PCM controller system. Architecture simulators have propelled the rapid development of computer architecture research in past decades, owing to its high productivity. Still, simulators of NVM do not appear to have rigorous and structured format due to a lack of predefined specifications, such as JEDEC and ONFI. In particular, PCM is one of the promising NVMs currently available in the market; however, its relating simulator is still architected as a standalone scheduler without any necessary module that ensures the reliability or throughput of the system. Furthermore, building such a simulator requires to be formatted concerning the coding style. In this chapter, we implement an all-inclusive PCM controller simulator (PCMCsim) that resembles the currently available product in the industry rather than a simple existing simulator while providing a guideline to program a cycle-accurate module, including contention behavior, with well-defined coding styles.

5.1 Motivation

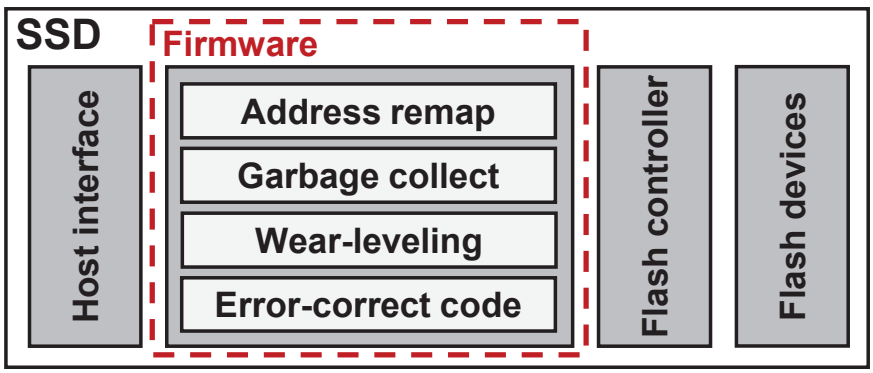


Figure 5.1: Architecture of a NAND-based SSD controller system

The necessity of an all-inclusive simulator. State-of-art memory simulators, such as NVMain [87], DRAMsim series [97], [99], and Ramulator [96], have been widely utilized for NVM-related research, including PCM itself. However, existing simulators cannot be directly used as PCM controller simulators because a PCM controller must consist of the modules enabling device management and performance enhancement. In particular, a PCM controller must translate host addressees to physical device addresses to prevent wear-out or WDE problems. In [91], it is announced that a PCM-based product, namely Intel Optane DCPMM, manages remapping mechanisms using an address indirection table (AIT) with a dedicated DRAM subsystem. Furthermore, PCM must compensate its higher latency than that of DRAM with an enlarged transaction unit, as discussed in Chapter 4; hence, an RMW module is also necessary for a PCM controller, which is also mentioned in [90], [91]. It is noteworthy that a NAND-based SSD controller has a similar command processing flow to that of PCM, which processes all commands with firmware (i.e., flash translation layer), as shown in Figure 5.1. In contrast, a PCM controller processes host commands through a hardware

processing path to maximize the host quality-of-service (QoS) [15]. In conclusion, we need to implement a simulator dedicated to the PCM controller that comprises the features mentioned above even though the publicized information is scarce.

The necessity of formatted programming models. An architecture simulator significantly improves the industry productivity, owing to its robustness compared to RTL models. However, the absence of formatted programming models in a simulator may lag the development progress for a novice. In particular, although event-driven simulators yield shorter simulation time compared to that of cycle-level simulators, event scheduling that simulates the concurrent behaviors in hardware must be managed manually. It is noteworthy that such management is not required in cycle-level simulators because cycle-level simulators try to simulate every cycle with a much longer simulation time than that of event-driven simulators, using a simple loop statement. Furthermore, event scheduling for the contention behavior between multiple masters also should be considered for cycle accuracy because the latency of a module generally originates from such ”bottleneck” points. In short, an event-driven PCM controller simulator that describes hardware contention behavior and provides the formatted programming models is required for future PCM-related research.

5.2 PCMCsim: PCM Controller Simulator

5.2.1 Architectural Overview

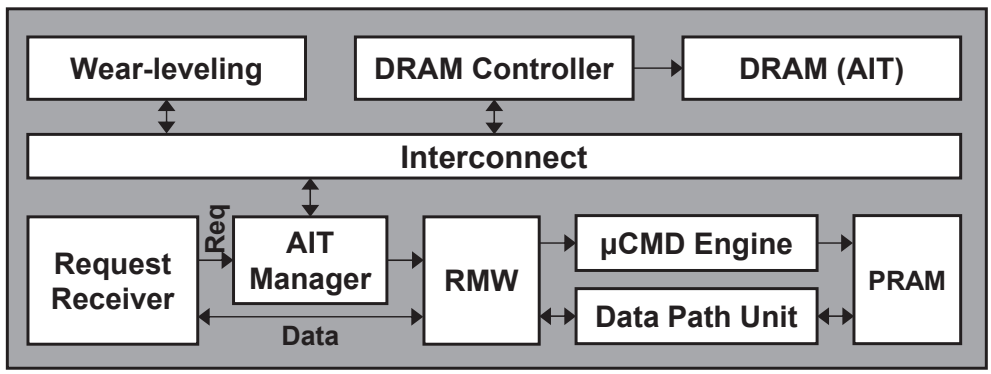


Figure 5.2: Architectural overview of PCMCsim.

Figure 5.2 shows the overall architecture of PCMCsim. All requests from the host processor first flow into a host interface, namely request receiver. For each request in the request receiver, a unique ID is allocated for performing out-of-order scheduling. After the ID allocation, requests get into AIT manager, which remaps a host address to a physical address. AIT manager first stacks requests in a buffer and obtains translation information stored in the AIT by accessing DRAM in the DRAM subsystem. Subsequently, translated requests flow into the RMW module, whereas the data of requests are migrated from the write data buffer in the request receiver to the data buffer in RMW. The RMW module reads as the transaction unit of PCM and writes the updated data back to the PCM (explained in Section 2.3.1). Both read and write requests are issued to the micro-command engine (uCMD engine) from the RMW module, which decomposes requests into micro-commands that can be recognized by PCM devices.

Finally, micro-commands are scheduled and issued to the PCM devices, where write data residing in data-path-unit (DPU) are signaled by issued micro-commands

and dispatched to the devices as well. For a read request, request ID is retired when the responded data goes back to the request receiver. For a write request, request ID is retired once uCMD engine is ready to issue the final micro command.

Please note that the wear-leveling algorithm is triggered from the RMW module when the metadata read from PCM reaches the predefined threshold. Thereafter, the remapping request containing remapping information is issued to the AIT manager for updating AIT in the DRAM subsystem.

5.2.2 Underlying Classes of PCMCsim

In this subsection, basic classes that comprise PCMCsim are explained in detail. Furthermore, the formatted programming style of a module in PCMCsim is also explained along with a toy example and intuitive pseudo-codes.

Component class

Component class is a parent class for all modules of PCMCsim that trigger event scheduling. This class can be considered as a wrapper of a hardware module that consists of basic functions for simulating various hardware behaviors. Followings are commonly used functions or variables in this class:

- *recvRequest( ), recvResponse( )*: these functions receive requests and responses from the connected modules, respectively.
- *isReady( )*: it checks the idleness of a connected module before issuing a command to that module.
- *ticks_per_cycle*: it is a variable that determines the frequency domain of this module. PCMCsim assumes the frequency of 1 THz if this variable is set to 1.

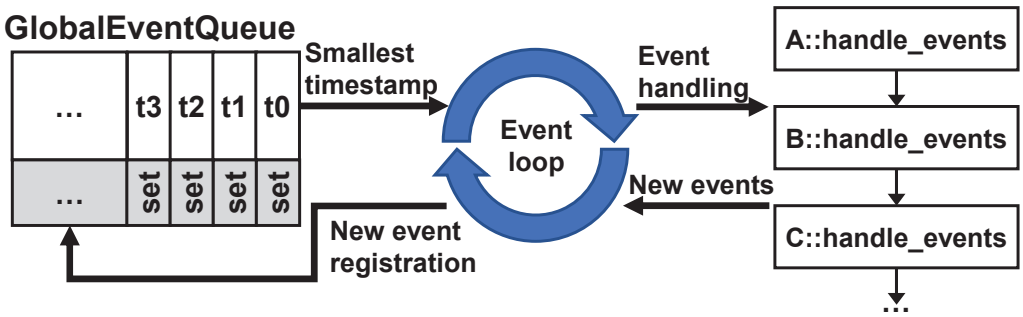


Figure 5.3: Illustration of event handling in PCMCsim.

In this chapter, we consider a ”tick” as 1 ps.

- *registerCallback( )*: this function is used for registering a callback function that handles the reserved event at a specific timing. This function is one of the essential functions in an event-driven simulator.

- *handle_events( )*: this function is a virtualized function that handles all registered callback functions at the current cycle. This function is automatically called from GlobalEventQueue class (see explanation in the following subsection), which is a centralized event management class of PCMCsim.

GlobalEventQueue class

GlobalEventQueue class is a centralized event management class of PCMCsim. In PCMCsim, all events are managed in a hierarchical manner, which highly utilizes the standard template library (STL) of C++, namely *map* and *set*. GlobalEventQueue class internally has a private object whose type is defined as $map\langle ncycle\_t,\ set\langle Component*\rangle\rangle$. This object is comprised of pairs of sets of event-driven modules and timestamps. For each timestamp, it indicates the moment that calls registered functions scheduled previously. For each component in $set\langle Component*\rangle$, a module that has registered events

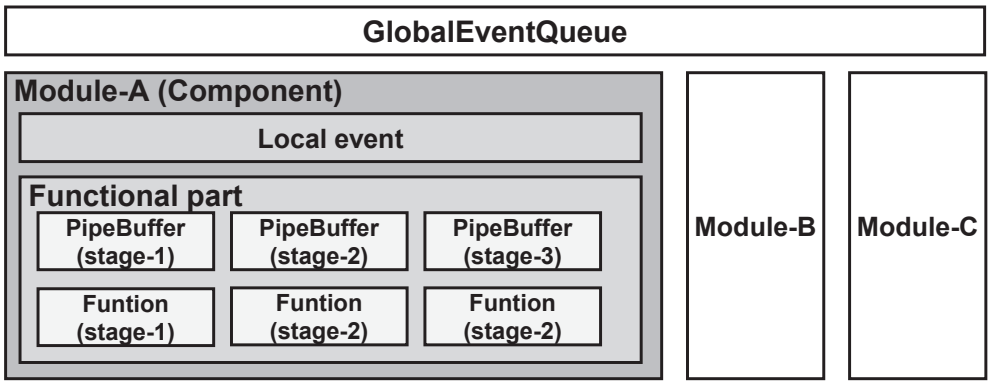


Figure 5.4: Case study system implemented with underlying classes of PCMCsim.

is included in this *set* data structure. The event handling process with this object is illustrated, as shown in Figure 5.3. As shown in the figure, the entry containing the smallest timestamp in the event queue object is popped within the loop statement in a member function of GlobalEventQueue. Subsequently, *handle_events( )* functions of all component modules bound by this timestamp are handled in order.

It is noteworthy that GlobalEventQueue class manages all timestamps with ”tick” resolution (i.e., 1 ps); hence, it is sufficient to instantiate one object of GlobalEventQueue class for handling events among several frequency domains.

PipeBuffer class

PipeBuffer class is a module for simulating pipeline registers in a hardware system. A pipeline is a common design strategy for enhancing hardware throughput with sequentially connected registers; that is, several operations can be executed concurrently in each pipelined register. Instances of PipeBuffer class can be connected to each other to form a pipelined hardware system. Furthermore, this class can automatically observe whether the pipelined register system is stalled. Subsequently, the content of the

Procedure: Module-A execution

```
/* Execute functions of the module */
stage-1_function( )
stage-2_function( )
stage-3_function( )

/* Proceed pipeline buffers */
proceed the contents in stage-1 pipeline buffer
proceed the contents in stage-2 pipeline buffer
proceed the contents in stage-3 pipeline buffer
output command = stage-3 pipeline buffer output

if output command is valid
  Module-B.recvRequest(output commd)

if a content exists in one of the pipeline buffers
  registerCallback(Module-A execution)
```

Figure 5.5: Pseudo-code for implementing the pipeline behavior of Module-A.

stalled register does not proceed to the next-stage register if the system is congested at the output. In conclusion, this high-level abstract class enables a formatted programming style for implementing user-defined modules.

Case study

Figure 5.4 shows an example system consisting of Component class, GlobalEventQueue class, and PipeBuffer class. As shown in the figure, Module-A has a three-stage

Procedure: Module-A/B execution

```
execute functions of Module-A or B
…


/* Register event of DPSRAM */
if last_wake_cycle == wake_cycle
  and free_cycle < current cycle:
  wake_cycle = current cycle + 1
  registerCallback(DPSRAM, wake_cycle)
```

(a)

Procedure: DPSRAM

```
if callee == Module-A:
  read SRAM contents
else if callee == Module-B:
  read SRAM contents

/* Update timing */
last_wake_cycle = wake_cycle
free_cycle = MAX(free_cycle, current+tRD)
registerCallback(DPSRAM, free_cycle)
```

(b)

Figure 5.6: Example pseudo-codes for implementing the contention behavior.

pipeline and is connected to Module-B. Local event in Module-A is a set of registered events, which are finally handled in *handle_events( )*. GlobalEventQueue behaves as a centralized event handling platform for Module-A, Module-B, and Module-C.

Figure 5.5 shows pseudo-code for implementing the pipeline behavior of Module-A. As shown in the figure, the functional part of each stage and the register proceeding procedure are completely decoupled. Furthermore, the output request at the final stage is issued to Module-B. If pipeline registers are checked as non-empty at the end of this function, it is scheduled again for completing the task. The simple pseudo-code shows that user-defined modules can be instantly implemented with underlying classes.

5.2.3 Implementation of Contention Behavior

Resource contention among multiple master modules is a common phenomenon in a hardware system. For example, a 2-input multiplexer is required if a DPSRAM is shared by two hardware modules, which is a generally applied design layout. Therefore, it is necessary to provide a formatted programming style concerning the con-

tention behavior. Figure 5.6 (a) and (b) show example pseudo-codes for implementing this behavior. The example assumes that a DPSRAM is shared by two modules, i.e., Module-A and Module-B. Figure 5.6 (a) shows the access process of each module, whereas Figure 5.6 (b) shows contention handling in the DPSRAM. For implementing the contention behavior, three variables are necessary, as shown in the figure:

- *wake_time*: this variable records the cycle moment when the function for accessing a shared resource (i.e., DPSRAM) is scheduled.
- *last_wake_time*: this variable records the cycle moment when the function for accessing a shared resource is actually called.
- *free_time*: this variable records the cycle moment with additive latency when the function is is called. This additive latency determines the access latency of a shared resource. In this example, tRD is used for indicating the read latency of DPSRAM.

In summary, we can formally implement the hardware contention behavior by simply leveraging three variables, which define attributes of a multiplexer for a shared resource, thereby improving the cycle accuracy of a simulator.

5.2.4 Modules of PCMCsim

This subsection explains implementation details of modules in PCMCsim that are briefly mentioned in Section 5.2.1. Please note that details of wear-leveling are not explained due to confidential issues. Furthermore, we explain the details of the JEDEC-based uCMD engine rather than the implementation used for correctness verification in Section 5.3.1 due to confidential issues as well.

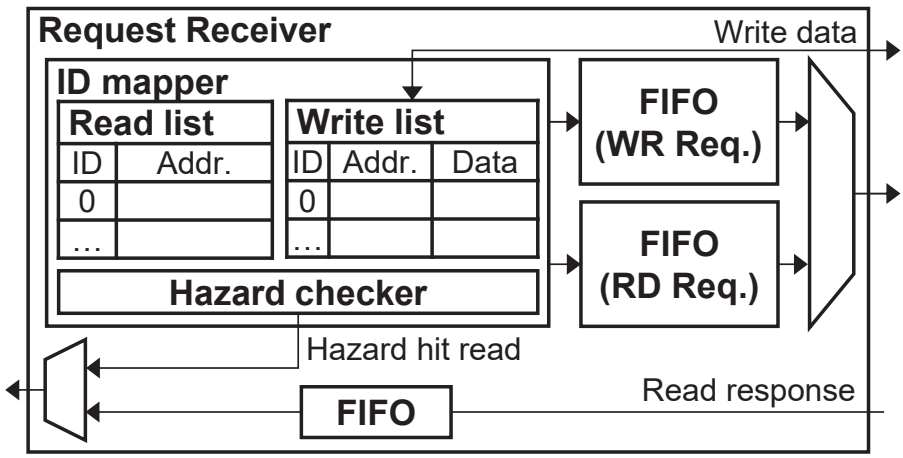


Figure 5.7: Implementation of request receiver.

Request Receiver

Figure 5.7 depicts the detailed architecture of request receiver. In particular, this module gets host requests and is controlled by two kinds of data structures:

- ID mapper: Since a PCM controller can schedule host requests in an out-of-order manner that is performed at the multiplexer at the output in the figure, ID must be granted for each request to respond to the host correctly. In this study, both read and write requests are separately managed for higher host QoS. Please note that the hazard checker in the ID mapper detects the cases of read-after-write and write-after-write and directly performs direct responses or updates, respectively.

- FIFO: In this module, three FIFOs are included. Two FIFOs at the right-hand side of the figure are for storing read and write requests, which are scheduled in an out-of-order manner. The final FIFO at the bottom of the figure is for storing read response data. Please note that a write request ID retires once the request and its data are dispatched to the next modules, namely AIT manager and RMW.

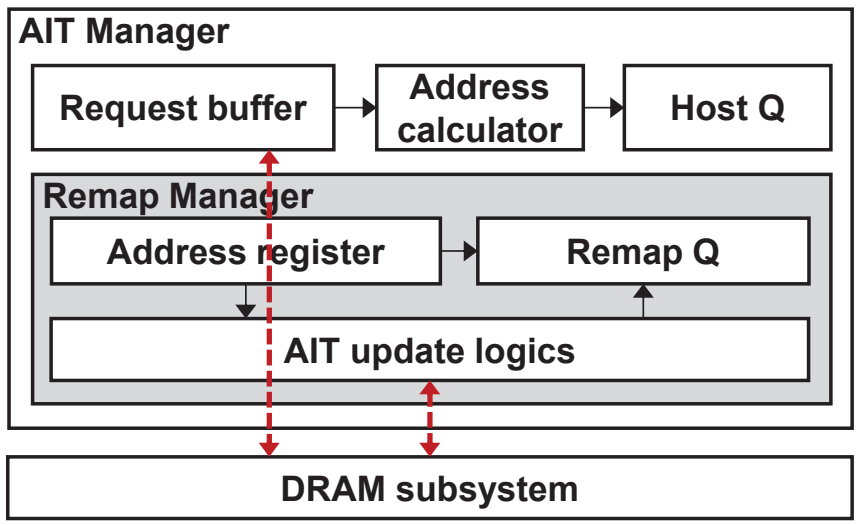


Figure 5.8: Implementation of AIT manager.

AIT Manager

AIT manager performs address translation for accessing PCM media devices. Figure 5.8 shows the detailed implementation of the AIT manager. The request comes from the request receiver first goes into the request buffer. The stacked requests in the request buffer are not issued directly, which generates new requests for obtaining physical address information from the DRAM subsystem, in which AIT is stored in DRAM rather than PCM for reducing the translation overhead. After the physical address information returns to the request buffer, the request goes into the host queue (i.e., Host Q) after address translation in the address calculator.

As shown in Figure 5.8, one more manager, called remap manager, is implemented for managing wear-leveling requests. A wear-leveling request comes from a micro-controller unit (MCU) block, as shown in Figure 5.2. The address register in Figure 5.8 stores addresses that are being remapped. Subsequently, it signals AIT update logic for generating requests for updating mapping information in AIT. After the corresponding

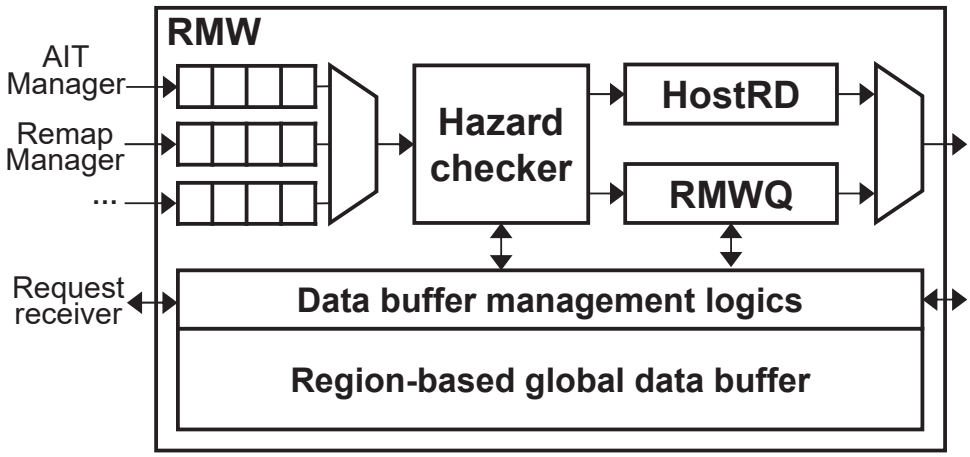


Figure 5.9: Implementation of RMW.

entries in AIT are updated, AIT update logic generates and pushes remapping requests into the remap queue (i.e., Remap Q).

It is noteworthy that significant performance overhead would be incurred by wear-leveling. This is because the remap manager is stalled until the request buffer and the Host Q and becomes empty for preventing data hazard between the host request and the remapping request. We leave the task of reducing overhead as future work.

RMW

As mentioned in Section 2.3.1, an RMW module generally reads data as the page size of the PCM device and responds or updates the partial data for the host processor. The implementation of RMW is illustrated in Figure 5.9. The RMW module processes requests as the following procedure:

- The RMW module gets requests from AIT manager and remap manager concurrently. If a write request comes into the module, the module obtains its data from the request receiver. It is noteworthy that PCMCsim implements further

input ports for the scalability of the module.

- One of the arbitrated requests is then passed to the hazard checker to check whether data having the same address exists in the global data buffer. If the address and data are found in the data buffer, data is directly responded to the host or updated according to the request type. Otherwise, the read request goes into the host-read register (i.e., HostRD), whereas the write request goes into RMW queues (i.e., a read queue for RMW and a queue for write-back).

- Finally, requests in HostRD and RMWQs are arbitrated and issued to the next module, namely the uCMD engine. HostRD has the highest priority for a higher host QoS, whereas the requests in RMWQs are arbitrated according to the watermark-based policy. The watermark-based policy assumes that RMW-read requests have higher priority than write requests, which are drained if the number of write requests overflows in the queue. Please note that the write data is signaled by its corresponding request when the request is ready to be issued. Subsequently, the data goes to DPU through data buffer management logic.

uCMD Engine

In this dissertation, we explain one of the child classes of the uCMD engine class that is implemented according to JEDEC DDR4 specification, as shown in Figure 5.10. An uCMD engine decomposes requests from the RMW module to micro-commands that are commonly used in a DDR4-based memory device, such as precharge and activation commands. According to the explained features, an uCMD engine also can be instantiated as a DRAM subsystem for AIT.

As shown in Figure 5.10, the implementation of the engine originates from a

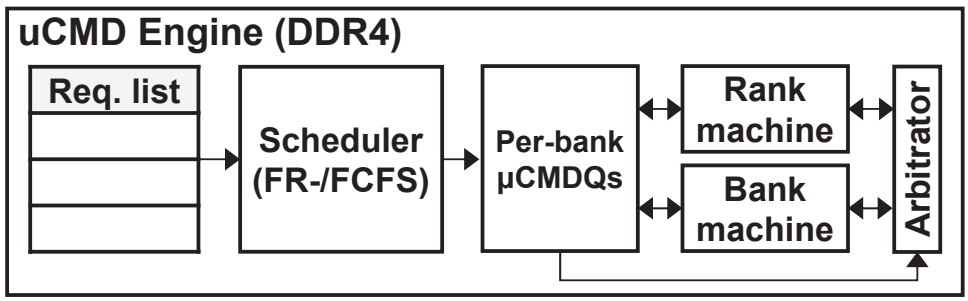


Figure 5.10: Implementation of uCMD engine.

practical DRAM controller IP design [127]. The requests in the request list are selected according to the scheduler policy (e.g., FR-FCFS policy for leveraging row hits in a device). According to its bank address, the selected request is decomposed into micro-commands and pushed into the corresponding micro-command queue (i.e., uCMDQ). Subsequently, issuable timings of commands in uCMDQs are monitored by two machines, called a rank machine and a bank machine. The former tracks rank and activation-specific timings for monitoring inter-/intra-rank accesses and four-activation window (FAW), respectively; the latter tracks other timing parameters to determine issuable commands according to the bank status. Concurrently, the arbitrator selects one uCMDQ with bank-first round-robin policy or rank-first round-robin policy, from which one command is issued at a time.

It is noteworthy that our uCMD engine supports both refresh operations (i.e., self-refresh and auto-refresh) and the power-down operation. In particular, the auto-refresh consists of the all-bank refresh, per-bank refresh, and fine-granularity refresh (FGR). For power-down mode, PCMCsim supports both fast-exit mode and slow-exit mode, which means it can be configured as DDR3 as well. Furthermore, the power model is also included in PCMCsim according to calculation functions explained in [128]. In conclusion, these features indicate that our PCMCsim can be used for simulating

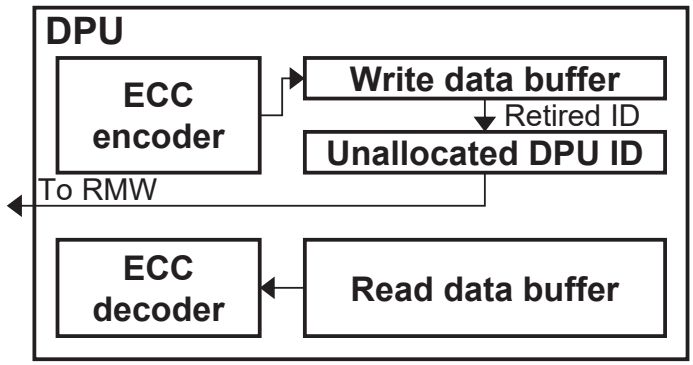


Figure 5.11: Implementation of DPU.

both PCM and DRAM with modified configurations; hence, the generalizability is guaranteed in the simulator for future research purposes.

DPU

DPU is a simple module that acts as a path for read data and write data from/to PCM devices, as shown in Figure 5.11. The figure has an unallocated DPU ID list that is used for conferring IDs for write data because the request information path and the data path into the uCMD engine and the DPU, respectively. Once the write command is ready to be issued to PCM, the uCMD engine instantly notifies DPU for dispatching the write data to PCM as well. Simultaneously, the retired DPU ID returns back to the unallocated DPU ID list for obtaining the next data from RMW. Please note that we reflect the multi-bit ECC feature to PCMCsim for reliability-related research (e.g., WDE) because PCM has lower reliability than DRAM.

5.3 Evaluation

5.3.1 Correctness of the Simulator

We have verified the functional correctness of our simulator against the traces extracted from the industrial RTL code, which implements the PCM controller based on the architecture of PCMCsim. We have implemented a macro that verifies the correspondence of commands between the output of PCMCsim and the output of RTL traces. Please note that the output is probed at each module in the PCM controller. Consequently, we have observed a functional accuracy of 99.5%. The slight discrepancy originates from the different input timings on a request list. Table 5.1 shows commands of RTL and PCMCsim at the request list in uCMD engine. The order discrepancy between RTL and PCMCsim occurs for last two commands, i.e., [0xD5, WRITE, 0x3300] and [0xCF, WRITE, 0x3480]. Different from RTL, PCMCsim shows that [0xD4, WRITE, 0x3480] lags behind [0xCF, WRITE, 0x3300]. This is because responses of corresponding RMW-READs of two WRITEs (i.e., 0x3300 and 0x3480)

Table 5.1: Commands of RTL and PCMCsim at the request list in uCMD engine

RTL				PCMCsim			
Request ID	Type	Address	Bank	Request ID	Type	Address	Bank
0xCB	WRITE	0x3200	9	0xCB	WRITE	0x3200	9
0xCF	RMW-READ	0x3300	9	0xCF	RMW-READ	0x3300	9
...				...			
0xD3	RMW-READ	0x3400	10	0xD3	RMW-READ	0x3400	10
0xD5	RMW-READ	0x3480	10	0xD5	RMW-READ	0x3480	10
...				...			
0xD5	WRITE	0x3480	10	0xCF	WRITE	0x3300	9
0xCF	WRITE	0x3300	9	0xD5	WRITE	0x3480	10

Table 5.2: Comparison with state-of-art memory simulators

Simulators	Sim. type	Standard	NVM features		Low-power	
			AIT	RMW	Self-refresh	Power-down
Ramulator [96]	Cycle-level	DDR4	×	×	○	○
DRAMsim3 [99]	Cycle-level	DDR4	×	×	○	×
NVMain [88]	Event-driven	DDR3	×	×	×	○
PCMCsim	Event-driven	DDR4	○	○	○	○

are interchanged in PCMCsim. In particular, we have found that the scheduling order of bank machine is 10 and 9 for both cases; however, bank-10 opened by [0xD3, RMW-READ, 0x3400] is closed after bank-9 opened by [0xCB, WRITE, 0x3200] is closed in PCMCsim, which means that the bank machine first schedules the command heading to bank-9 (i.e., [0xCF, RMW-READ, 0x3300]). The reason for this phenomenon is the lack of cycle accuracy incurred by I/O buffer latency in RTL, which is unnecessary in a simulator. Please note that all discrepancies stem from the same reason.

5.3.2 Comparison with Other Simulators

Table 5.2 shows the feature comparison with other simulators, namely DRAMsim3 and NVMain. The first column shows the simulation type of the simulator. Since DRAMsim3 is a cycle-level simulator, it is the slowest among all simulators. We have observed that PCMCsim is 16× faster than DRAMsim3 by executing traces of SPEC CPU 2017 applications, whereas a similar simulation time is obtained compared to NVMain. The second column shows the JEDEC standard that each simulator supports, in which PCMCsim is the first NVM simulator that supports DDR4 specification compared to NVMain. The third column shows that only PCMCsim supports necessary features for the next-generation NVM-based controller (i.e., AIT subsystem

and RMW). Finally, the fourth column shows that DRAMsim and NVMain support only one of the low-power operations. In particular, NVMain provides configurable timing concerning self-refresh; however, the actual function behavior is not provided in the simulator. In contrast, both operations are implemented in PCMCsim, thereby enhancing the scalability of the simulator.

5.4 Summary

PCM is a promising byte-addressable NVM; however, no simulator that well describes essential features for a PCM controller is available in the academic research area. This chapter compiles all these features into one simulator, PCMCsim, including hardware processing path, AIT that includes a complete DRAM subsystem, RMW, and ECC. Furthermore, we provide a guideline for programming user-defined modules formally as well as the resource contention behavior to enhance the research productivity. We also implement features of DRAM; hence, directly instantiating a DRAM-based system is possible without incorporating external DRAM simulators. PCMCsim yields the functional accuracy of 99.5% with 16× simulation time reduction compared to a state-of-art simulator.

Chapter 6

Conclusion

The advent of applications having a tremendous memory footprint, such as in-memory databases or deep learning, leads to increasing demand for large-capacity main memory. As the DRAM process technology reaches its limitation, PCM becomes a promising candidate as next-generation memory owing to its attractive physical characteristics, particularly its high scalability and endurance. Still, PCM-related technologies in the industry stagnate due to three problems: reliability problems, higher latency issues than DRAM, and the absence of a simulation platform that incorporates the characteristics of a PCM controller.

In this dissertation, we have completed mainly three tasks to overcome the problems mentioned above and contribute to a PCM-based main memory system. First of all, a WDE mitigation scheme that introduces table-based management within a PCM product (or module) is proposed. This is the first approach that leverages an announced process parameter to restore WDE-vulnerable cells on demand. Secondly, we first consider a more practical PCM controller architecture that incorporates an RMW module.

A simple RMW module significantly degrades the performance of a PCM-based system that runs normal CPU applications. Therefore, we leverage a part of the DRAM region that already exists in a practical PCM-based product to enhance the overall performance notably, including the read reliability. Lastly, we gather information necessary for a practical PCM-based product, such as the AIT subsystem and the RMW module, to build an all-inclusive PCM controller simulator. The proposed simulator is verified against industrial RTL traces with high functional accuracy. Furthermore, our simulator is flexibly configurable as both PCM and DRAM, which is suitable for broader memory-related research in the future.

In conclusion, three novel contributions would establish a foundation for further development of PCM-related technology and propel the popularization of PCM-related products. This dissertation takes several practical aspects of a PCM controller into consideration as much as possible. Therefore, we firmly believe that our work would be helpful to both industry and academic research with growing interests in PCM.

Bibliography

[1] Memcached. (2018). “About memcached,” [Online]. Available: https://memcached.org/about.

[2] Redis. (-). “An introduction to redis data types and abstractions,” [Online]. Available: https://redis.io/topics/data-types-intro.

[3] SAP. (2020). “What is sap hana?” [Online]. Available: https://www.sap.com/products/hana.html.

[4] H. Jin, B. Liu, W. Jiang, Y. Ma, X. Shi, B. He, and S. Zhao, “Layer-centric memory reuse and data migration for extreme-scale deep learning on many-core architectures,” *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, Sep. 2018.

[5] KSIA, “Ddr5 시대의 도래,” *Silicon TIMES*, Oct. 2020.

[6] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, “Phase-change technology and the future of main memory,” *IEEE Micro*, vol. 30, no. 1, pp. 143–143, 2010.

[7] J. Meena, S. Sze, U. Chand, and T.-Y. Tseng, “Overview of emerging non-volatile memory technologies,” *Nanoscale Research Letters*, vol. 9, pp. 1–33, Sep. 2014.

[8] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high-performance computing," *Computing in Science Engineering*, vol. 17, no. 2, pp. 73–82, 2015.

[9] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.

[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009, pp. 2–13.

[11] S. Mittal, "A survey of soft-error mitigation techniques for non-volatile memories," *MDPI Computers*, vol. 6, no. 1, 2017.

[12] S. Sundararaman, N. Talagala, D. Das, A. Mudrankit, and D. Arteaga, "Towards software defined persistent memory: Rethinking software support for heterogeneous memory architectures," in *Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads (INFLOW)*, 2015, 6:1–6:10.

[13] J. Guerra, L. Marmol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei, "Software persistent memory," in *2012 USENIX Annual Technical Conference*, 2012, pp. 22–29.

[14] Intel. (2019). "Intel memory drive technology set up and configuration guide," [Online]. Available: `https://www.intel.com/content/dam/support/us/en/documents/memory-and-storage/intel-mdt-setup-guide.pdf`.

[15] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3d xpoint technology," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1822–1833, 2017.

[16] 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 *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011, pp. 105–118.

[17] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "Atom: Atomic durability in non-volatile memory through hardware logging," in *Proceedings of the 23rd International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 361–372.

[18] J. Jeong, C. H. Park, J. Huh, and S. Maeng, "Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory," in *Proceedings of the 51st International Symposium on Microarchitecture (MICRO)*, Oct. 2018, pp. 520–532.

[19] C.-H. Lai, J. Zhao, and C.-L. Yang, "Leave the cache hierarchy operation as it is: A new persistent memory accelerating approach," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, New York, NY, USA: ACM, 2017, 5:1–5:6.

[20] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proceedings of the 46th International Symposium on Microarchitecture (MICRO)*, Dec. 2013, pp. 421–432.

[21] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the 9th European Conference on Computer Systems (EuroSys)*, 2014, 15:1–15:15.

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

[23] S. Shin, J. Tuck, and Y. Solihin, "Proteus: A flexible and fast software supported hardware logging approach for nvm," in *Proceedings of the 50th Annual International Symposium on Microarchitecture (MICRO)*, 2017, pp. 178–190.

[24] J. Choe. (). "Intel 3d xpoint memory die removed from intel optane pcm (phase change memory)," [Online]. Available: `https://www.techinsights.com/blog/intel-3d-xpoint-memory-die-removed-intel-optanetm-pcm-phase-change-memory`.

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

[26] J. Yang, B. Li, and D. J. Lilja, "Exploring performance characteristics of the optane 3d xpoint storage technology," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 5, no. 1, Feb. 2020.

[27] K. Wu, A. Arpaci-Dusseau, R. Arpaci-Dusseau, R. Sen, and K. Park, "Exploiting intel optane ssd for microsoft sql server," in *Proceedings of the 15th International Workshop on Data Management on New Hardware*, New York, NY, USA: Association for Computing Machinery, 2019.

[28] V. Mironov, A. Kudryavtsev, Y. Alexeev, A. Moskovsky, I. Kulikov, and I. Chernykh, "Evaluation of intel memory drive technology performance for scientific applications," in *Proceedings of the Workshop on Memory Centric High Performance Computing*, New York, NY, USA: Association for Computing Machinery, 2018, pp. 14–21.

[29] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," in *12th USENIX Conference on File and Storage Technologies (FAST 14)*, Santa Clara, CA: USENIX Association, Feb. 2014, pp. 33–45.

[30] M. Weiland, H. Brunst, T. Quintino, N. Johnson, O. Iffrig, S. Smart, C. Herold, A. Bonanni, A. Jackson, and M. Parsons, "An early evaluation of intel¡?s optane dc persistent memory module and its impact on high-performance scientific applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA: Association for Computing Machinery, 2019.

[31] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a dram-nvm hybrid memory architecture for hpc applications using intel optane dc persistent memory modules," in *Proceedings of the International Symposium on Memory Systems*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 288–303.

[32] I. B. Peng, M. B. Gokhale, and E. W. Green, "System evaluation of the intel optane byte-addressable nvm," in *Proceedings of the International Symposium on Memory Systems*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 304–315.

[33] 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, 2011.

[34] 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 *Proceedings of the 42nd Annual International Symposium on Microarchitecture (MICRO)*, 2009, pp. 14–23.

[35] J. Yun, S. Lee, and S. Yoo, "Dynamic wear leveling for phase-change memories with endurance variations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1604–1615, 2015.

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

[37] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *Proceeding of the 2007 IEEE International Symposium on Circuits and Systems*, May 2007, pp. 3014–3017.

[38] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *Proceedings of the 42nd Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2009, pp. 347–357.

[39] 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 *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 1081–1090.

[40] 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 *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 519–531.

[41] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing pcm main memory lifetime," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2010, pp. 914–919.

[42] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically replicated memory: Building reliable systems from nanoscale resistive memories," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, New York, NY, USA: Association for Computing Machinery, 2010, pp. 3–14.

[43] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, New York, NY, USA: Association for Computing Machinery, 2010, pp. 141–152.

[44] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H. S. Lee, "Safer: Stuck-at-fault error recovery for memories," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2010, pp. 115–124.

[45] M. K. Qureshi, "Pay-as-you-go: Low-overhead hard-error correction for phase change memories," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 318–328.

[46] J. Wang, X. Dong, and Y. Xie, "Point and discard: A hard-error-tolerant architecture for non-volatile last level caches," in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, 2012, pp. 253–258.

[47] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," in *Proceedings of the 46th Annual International Symposium on Microarchitecture (MICRO)*, 2013, pp. 433–444.

[48] M. K. Tavana, A. K. Ziabari, M. Arjomand, M. Kandemir, C. Das, and D. Kaeli, "Remap: A reliability/endurance mechanism for advancing pcm," in *Proceedings of the International Symposium on Memory Systems*, New York, NY, USA: Association for Computing Machinery, 2017, pp. 385–398.

[49] J. Zhang, D. Kline, L. Fang, R. Melhem, and A. K. Jones, "Dynamic partitioning to mitigate stuck-at faults in emerging memories," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 651–658.

[50] S. Swami, P. M. Palangappa, and K. Mohanram, "Ecs: Error-correcting strings for lifetime improvements in nonvolatile memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, no. 4, 2017.

[51] Y. Kim, S. Yoo, and S. Lee, "Write performance improvement by hiding r drift latency in phase-change ram," in *DAC Design Automation Conference 2012*, 2012, pp. 897–906.

[52] W. Zhang and T. Li, "Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system," in *Proceedings of the International Conference on Dependable Systems and Networks*, Jul. 2011, pp. 197–208.

[53] W. Xu and T. Zhang, "A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1357–1367, 2011.

[54] D. H. Yoon, J. Chang, R. S. Schreiber, and N. P. Jouppi, "Practical nonvolatile multilevel-cell phase change memory," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[55] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *Proceedings of the 18th International Symposium on High-Performance Computer Architecture (HPCA)*, 2012, pp. 1–12.

[56] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, New York, NY, USA: Association for Computing Machinery, 2013, pp. 440–451.

[57] M. Imran, T. Kwon, J. M. You, and J. Yang, "Flipcy: Efficient pattern redistribution for enhancing mlc pcm reliability and storage density," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[58] R. Wang, L. Jiang, Y. Zhang, and J. Yang, "Sd-pcm: Constructing reliable super dense phase change memory under write disturbance," in *Proceedings of the 20th International Conference on Architectural Support for Programming*

Languages and Operating Systems (ASPLOS), New York, NY, USA: ACM, 2015, pp. 19–31.

[59] L. Jiang, Y. Zhang, and J. Yang, "Mitigating write disturbance in super-dense phase change memories," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2014, pp. 216–227.

[60] S. Swami and K. Mohanram, "Adam: Architecture for write disturbance mitigation in scaled phase change memory," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2018, pp. 1235–1240.

[61] M. Imran and J.-S. Y. T. Kwon, "Effective write disturbance mitigation encoding scheme for high-density pcm," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[62] J. Jang, W. Shin, J. Choi, Y. Kim, and L. Kim, "Sparse-insertion write cache to mitigate write disturbance errors in phase change memory," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 752–764, May 2019.

[63] R. Wang, L. Jiang, Y. Zhang, L. Wang, and J. Yang, "Exploit imbalanced cell writes to mitigate write disturbance in dense phase change memory," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, Jun. 2015, pp. 1–6.

[64] S. H. Lee, "Method of driving phase change memory device capable of reducing heat disturbance," 0 204 664, 2014.

[65] 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 *IEEE 8th International Memory Workshop (IMW)*, 2016, pp. 1–4.

[66] U. Russo, D. Ielmini, A. Redaelli, and A. L. Lacaita, "Modeling of programming and read performance in phase-chanage memories-part i: Cell optimization and scaling," *IEEE Transactions on Electron Devices*, vol. 55, no. 2, pp. 506–514, Feb. 2008.

[67] U. Russo, D. Ielmini, A. Redaelli, and A. L. Lacaita, "Modeling of programming and read performance in phase-change memories-part ii: Program disturb and mixed-scaling approach," *IEEE Transactions on Electron Devices*, vol. 55, no. 2, pp. 515–522, Feb. 2008.

[68] P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi, "Reducing read latency of phase change memory via early read and turbo read," in *Proceedings of the 21st International Symposium on High-Performance Computer Architecture (HPCA)*, 2015, pp. 309–319.

[69] U. Russo, D. Ielmini, and A. L. Lacaita, "Analytical modeling of chalcogenide crystallization for pcm data-retention extrapolation," *IEEE Transactions on Electron Devices*, vol. 54, no. 10, pp. 2769–2777, 2007.

[70] U. Russo, D. Ielmini, A. Redaelli, and A. L. Lacaita, "Intrinsic data retention in nanoscaled phase-change memories-part i: Monte carlo model for crystallization and percolation," *IEEE Transactions on Electron Devices*, vol. 53, no. 12, pp. 3032–3039, Dec. 2006.

[71] A. Redaelli, D. Ielmini, U. Russo, and A. L. Lacaita, "Intrinsic data retention in nanoscaled phase-change memories-part ii: Statistical analysis and prediction of failure time," *IEEE Transactions on Electron Devices*, vol. 53, no. 12, pp. 3040–3046, Dec. 2006.

[72] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceedings of the 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[73] O. Mutlu and J. S. Kim, *Rowhammer: A retrospective*, 2019. arXiv: 1904.09724 [cs.CR].

[74] M. Kim, J. Choi, H. Kim, and H. Lee, "An effective dram address remapping for mitigating rowhammer errors," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1428–1441, 2019.

[75] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "Twice: Preventing rowhammering by exploiting time window counters," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 385–396.

[76] 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 *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*, 2012, pp. 380–391.

[77] P. Zhou, J. Y. B. Zhao, and Y. Zhang, "Throughput enhancement for phase change memories," *IEEE Transactions on Computers*, vol. 63, pp. 2080–2093, 2014.

[78] F. Xia, D. Jiang, J. Xiong, and N. Sun, "Hikv: A hybrid index key-value store for dram-nvm memory systems," in *2017 USENIX Annual Technical Conference*, 2017, pp. 349–362.

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

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

[81] R. Fisher, "Optimizing nand flash performance," in *Flash Memory Summit*, 2008.

[82] 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 *2011 Symposium on VLSI Circuits - Digest of Technical Papers*, 2011, pp. 196–197.

[83] J. Kim, S. Kim, and J. Kim, "Subpage programming for extending the lifetime of nand flash memory," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 555–560.

[84] 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 *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[85] Intel, *ECC Handling Issues on Intel XScale I/O Processors*, 1st ed. 2003.

[86] B. Beeler. (2019). "Intel optane dc persistent memory module (pmm)," [Online]. Available: https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm.

[87] M. Poremba and Y. Xie, “Nvmain: An architectural-level main memory simulator for emerging non-volatile memories,” in *2012 IEEE Computer Society Annual Symposium on VLSI*, Aug. 2012, pp. 392–397.

[88] M. Poremba, T. Zhang, and Y. Xie, “Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems,” *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[89] S. Chen, P. B. Gibbons, and S. Nath, “Rethinking database algorithms for phase change memory,” in *5th Biennial Conference on Innovative Data Systems Research, Conference Proceedings (CIDR)*, Apr. 2011, pp. 21–31.

[90] H. Lee, M. Kim, H. Kim, H. Kim, and H.-J. Lee, “Integration and boost of a read-modify-write module in phase change memory system,” *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1772–1784, 2019.

[91] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson. (2019). “Basic performance measurements of the intel optane DC persistent memory module.” arXiv: `1903.05714`, [Online]. Available: `http://arxiv.org/abs/1903.05714`.

[92] J. H. Edmondson, R. A. Alfieri, M. F. Harris, and S. E. Molnar, “Coalescing to avoid read-modify-write during compressed data operations,” 9 058 792, 2015.

[93] Intel, “Write combining memory implementation guidelines,” 1998.

[94] Z. Deng, L. Zhang, N. Mishra, H. Hoffmann, and F. T. Chong, “Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in nvms,” in *Proceedings of the 50th Annual IEEE/ACM International*

Symposium on Microarchitecture (MICRO), New York, NY, USA: Association for Computing Machinery, 2017, pp. 232–244.

[95] H. Kim, H. Park, T. Kim, K. Cho, E. Lee, S. Ryu, H.-J. Lee, K. Choi, and J. Lee, "Gradpim: A practical processing-in-dram architecture for gradient descent," in *Proceedings of the 27th Annual IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[96] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[97] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[98] K. Doshi, E. Giles, and P. Varman, "Atomic persistence for scm with a non-intrusive backend controller," in *Proceedings of the 22nd International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 77–89.

[99] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[100] JEDEC, "Jesd79-3 ddr3 sdram standard," Jun. 2007.

[101] JEDEC, "Jesd79-4 ddr4 sdram standard," Sep. 2012.

[102] H. Lee, S. Lee, M. Kim, H. Kim, H. Kim, and H.-J. Lee, "Imdb: A low-cost in-module disturbance barrier for mitigating write disturbance errors in phase-change memory," in *Proceedings of the 57th Annual Design Automation Conference (DAC) WIP*, 2020.

[103] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, Feb. 2020.

[104] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure pcm systems by online detection of malicious write streams," in *Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 478–489.

[105] Hewlett Packard Lab. (). "Cacti 6.5," [Online]. Available: `https://www.hpl.hp.com/research/cacti/`.

[106] M. N. Bojnordi and E. Ipek, "Pardis: A programmable memory controller for the ddrx interfacing standards," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 13–24.

[107] Mentor-Graphics. (2013). "Ddr4 and lpddr4 - broad design verification and challenges," [Online]. Available: `https://www.mentor.com/pcb/multimedia/player/ddr4-and-lpddr4-board-design-verification-and-challenges-356bbc16-6195-4d78-ba85-5496362bec44`.

[108] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 994–1007, Jul. 2012.

[109] (). "Arm cortex a-15 (samsung exynos 5250)," [Online]. Available: `https://www.7-cpu.com/cpu/Cortex-A15.html`.

[110] T. Instruments. (). “Reliability terminology,” [Online]. Available: https://www.ti.com/support-quality/reliability/reliability-terminology.html.

[111] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, “The soft error problem: An architectural perspective,” in *Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA)*, 2005, pp. 243–247.

[112] C. Yang, Y. Emre, Y. Cao, and C. Chakrabarti, “Improving reliability of non-volatile memory technologies through circuit level techniques and error control coding,” *EURASIP Journal on Advances in Signal Processing*, vol. 34, no. 211, Oct. 2012.

[113] D. Strukov, “The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories,” in *2006 40th Asilomar Conference on Signals, Systems and Computers*, 2006, pp. 1183–1187.

[114] A. Abel and J. Reineke, “Reverse engineering of cache replacement policies in intel microprocessors and their evaluation,” in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 141–142.

[115] P. Vila, P. Ganty, M. Guarnieri, and B. Köpf, “Cachequery: Learning replacement policies from hardware caches,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA: Association for Computing Machinery, 2020, pp. 519–532.

[116] Y. Zhang, D. Feng, W. Tong, Y. Hua, J. Liu, Z. Tan, C. Wang, B. Wu, Z. Li, and G. Xu, “Cacf: A novel circuit architecture co-optimization framework for

improving performance, reliability and energy of reram-based main memory system," vol. 15, no. 2, May 2018.

[117] Y. Zhang, D. Feng, J. Liu, W. Tong, B. Wu, and C. Fang, "A novel reram-based main memory structure for optimizing access latency and reliability," in *Proceedings of the 54th Annual Design Automation Conference 2017*, New York, NY, USA: Association for Computing Machinery, 2017.

[118] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 476–488.

[119] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in mlc phase change memory," in *Proceedings of the 18th International Symposium on High Performance Computer Architecture (HPCA)*, 2012, pp. 201–210.

[120] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA)*, 2010, pp. 1–11.

[121] P. R. Hiller, W. P. Hovis, and J. A. Kirscht, "Memory controller and method for optimized read/modify/write performance," 7 328 317, 2008.

[122] F. Zeng, L. Qiao, M. Liu, and Z. Tang, "Memory performance characterization of spec cpu2006 benchmarks using tsim," *Physics Procedia*, vol. 33, pp. 1029–1035, 2012.

[123] J. Zhao, O. Mutlu, and Y. Xie, "Firm: Fair and high-performance memory control for persistent memory systems," in *Proceedings of the 47th Annual International Symposium on Microarchitecture (MICRO)*, 2014, pp. 153–165.

[124] Intel. (2010). "Microprocessor power impacts," [Online]. Available: `https://www.glsvlsi.org/archive/glsvlsi10/pant-GLSVLSI-talk.pdf`.

[125] M. Poremba, T. Zhang, and Y. Xie, "Fine-granularity tile-level parallelism in non-volatile memory architecture with two-dimensional bank subdivision," in *Proceedings of the 53th Annual Design Automation Conference (DAC)*, 2016, pp. 1–6.

[126] Y. Du, M. Zhou, B. R. Childers, D. Mosse, and R. Melhem, "Bit mapping for balanced pcm cell programming," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 428–439.

[127] Xilinx, "Virtex-6 fpga memory interface solutions user guide," Mar. 2011.

[128] Micron, "Tn-40-07: Calculating memory power for ddr4 sdram," 2018.

초 록

상변화 메모리는(PCM) 매력적인 특성을 통해 메모리 시스템의 새로운 시대의 시작을 알렸다. 많은 메모리 관련 제품 제조업체(예 : 인텔, SK 하이닉스, 삼성)가 관련 제품 개발에 박차를 가하고 있다. PCM은 단순히 대규모 데이터베이스에만 국한되지 않고 다양한 상황에 적용될 수 있다. 예를 들어, PCM은 비휘발성으로 인해 대기 전력이 낮다. 따라서 계산 집약적인 애플리케이션 또는 모바일 애플리케이션은(즉, 긴 메모리 유휴 시간) PCM 기반 컴퓨팅 시스템에서 실행하기에 적합하다.

PCM의 이러한 매력적인 특성에도 불구하고 PCM은 낮은 신뢰성과 긴 대기 시간으로 인해 여전히 일반 산업 시장에서는 DRAM과 다소 격차가 있다. 특히 낮은 신뢰성은 지난 수십 년 동안 PCM 기술의 발전을 저해하는 문제다. 반도체 공정 기술이 수년에 걸쳐 빠르게 축소됨에 따라 DRAM은 10nm 급 공정 기술에 도달하였다. 이어서, 쓰기 방해 오류 (WDE)가 54nm 등급 프로세스 기술 아래로 축소되면 PCM에 심각한 문제가 될 것으로 보고되었다. 따라서, WDE 문제를 해결하는 것은 PCM이 DRAM과 동등한 경쟁력을 갖추도록 하는 데 있어 필수적이다. 이 문제를 극복하기 위해 이 논문에서는 2-레벨 SRAM 기반 테이블을 활용하여 WDE 수를 크게 줄여 필요에 따라 준 안정 셀을 복원할 수 있는 새로운 접근 방식을 제안한다. 또한, 원래 SRAM에서 수백 개의 읽기 포트가 필요한 대체 정책을 구현하기 위해 새로운 랜덤 기반의 기법을 제안한다.

PCM의 두 번째 문제는 DRAM에 비해 지연 시간이 길다는 것이다. 특히 PCM은 더 큰 트랜잭션 단위를 채택하여 단위시간 당 데이터 처리량 향상을 도모한다. 그러나 범용 프로세서 캐시 라인과 다른 유닛 크기는 읽기-수정-쓰기 (RMW) 모듈의 도입으로 인해 시스템 성능을 저하하게 된다. PCM 기반 메모리 시스템에서 RMW 관련 연구가 없었기 때문에 본 논문은 RMW 모듈을 탑재 한 PCM 기반 메모리 시스템의 전반적인 시스템 성능과 신뢰성을 향상하게 시킬 수 있는 새로운 아키텍처를 제안한다. 제안된 아키텍처는 추가 스토리지 리소스를 도입하지 않고도 데이터 재사용성을 향상시킨다. 또한, 성능 향상을 위해 명령 유형과 관계없이 명령을 병합하는 새로운 작업을 제안한다.

또 다른 문제는 PCM을 위한 완전한 시뮬레이션 플랫폼이 부재하다는 것이다. PCM 관련 제품(예 : Intel Optane)에 대해 발표된 정보는 대외비 문제로 인해 부족하다. 하지만 알려져 있는 정보를 적절히 취합하면 시중 제품과 유사한 아키텍처 시뮬레이터를 개발할 수 있다. 이를 위해 본 논문은 PCM 메모리 컨트롤러에 필요한 모든 모듈 정보를 활용하여 향후 이와 관련된 연구에서 충분히 사용 가능한 전용 시뮬레이터를 구현하였다.

주요어: 컴퓨터 아키텍처, 비휘발성 메모리, 상변화 메모리, 쓰기 간섭, 읽기 간섭, 읽기-수정-쓰기 모듈, 메모리 시뮬레이터

학번: 2016-27167

ACKNOWLEDGMENT

While wrapping up the dissertation, I would like to express sincere appreciation to my advisor, Professor Hyuk-Jae Lee, for his supports and encouragement. He always provides me priceless advice concerning both academic knowledge and personality. I also want to thank committee members of my dissertation for their suggestions: Professor Deog-Kyoon Jeong, Professor Jangwoo Kim, Professor Jaewoong Sim, and Vice President Soojung Ryu at SK Telecom. In addition, I would like to express gratitude to Professor Hyun Kim at Seoul National University of Science and Technology and Professor Sunwoong Kim at the University of Washington. Professor Hyun Kim always provides me invaluable suggestions for my research and careers with his rigorous insights. Professor Sunwoong Kim was one of the mentors when I was a freshman, during which I have caught a lot about research methodologies from him.

Furthermore, I want to thank companies in CAPP. Special appreciations go to Jinwoo Park, Hyunmin Jung, Moonsoo Kim, Woojae Shin, Boyeal Kim, Seungyong Lee, Shinwoo Kang, Byeongki Song, Seokbo Shim, Hyeonggi Seong, and Hyungsuk Kim, with whom I have worked. In addition, I want to appreciate Donghyeon Lee and Kwangsoo Lee. I could not have joined CAPP and started Ph.D. work without their help during the internship. Finally, I would like to express gratitude to my family, particularly my parents, for their ceaseless supports and self-sacrificing contributions. I could not have obtained a Ph.D. degree without them.