공학박사학위논문

Performance Improvement of Memory System with LPDDR2-NVM

LPDDR2-NVM 기반 메모리 시스템의 성능 개선

2015년 2월

서울대학교 대학원
전기·컴퓨터공학부
박 재 현

Performance Improvement of Memory System with LPDDR2-NVM

LPDDR2-NVM 기반 메모리 시스템의 성능 개선

지도교수 염헌영

이 논문을 공학박사학위논문으로 제출함

2014년 12월

서울대학교 대학원

전기·컴퓨터공학부

박 재 현

박재현의 박사학위논문을 인준함

2014년 12월

위 원 장 하 순 회 (인)

부위원장 염 헌 영 (인)

위 원 최 기 영 (인)

위 원 장 래 혁 (인)

위 원 이 형 규 (인)

Abstract

Typical memory systems have used a synchronous random access memory (SRAM), a dynamic random access memory (DRAM), and a NAND flash in a cache, main memory, and storage, respectively. However, these traditional memory devices have limitations such as volatility, low density, and high leakage power. Therefore, emerging non-volatile memory (NVM) technologies such as phase change memory (PCM), spin-torque transfer random access memory (STT-RAM), and resistive random access memory (RRAM) are considered as an alternative of traditional memory devices due to its non-volatility, high density, and low-power. These numerous benefits of emerging NVMs motivate researchers to investigate the adoption of NVMs to the memory hierarchy.

Low power double data rate 2 non-volatile memory (LPDDR2-NVM) has been deemed the standard interface to connect NVMs because the characteristics of emerging NVMs are different to the traditional memory devices. The operation of LPDDR2-NVM is not same as the conventional DRAM, but most of the previous literature does not consider or overlook this standard interface.

This dissertation proposes system-level optimization methods to maximize the performance of memory system with LPDDR2-NVM. To this end, we first implement an LPDDR2-NVM prototype to extract parameters of memory system, and then we implement

a system-level simulator that reflects the realistic parameters. Second, we analyze the effect of row buffer architecture on the performance of the memory system though the intensive evaluation. Based on clues from evaluation, we propose a system-level method that improves performance memory system by reforming the way of interfacing LPDDR2-NVM. We also present the limitation of static row buffer architecture and propose a system-level method that mimics reconfigurable row buffer architecture.

Keywords: LPDDR2-NVM, memory system, performance optimization

Student number: 2006-23166

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

For several decades, the density of transistor and operation frequency increase as process technology shrinks. However, the power wall, seemingly intractable obstacle until now, has led architects to explore the instruction-level parallelism (ILP) in the last few years. The results of this paradigm shift marked the beginning of the multicore era. Chip Multiprocessors (CMPs) now employ arrays of lightweight processing cores. This abundance of on-chip computational resources puts an enormous strain on the memory systems. The desire to feed the beast necessitates both higher memory bandwidth and increased memory capacities.

In the typical memory systems, synchronous random access memory (SRAM), dynamic random access memory (DRAM), and NAND flash have been used as main components of the memory systems. SRAM operates at high frequency while it struggles under the low density and high leakage power due to its intrinsic cell structure. DRAM has a high read/write performance, relatively large capacity where as it requires refresh opera-

tions which consume significant energy to keep the data. Moreover, DRAM fails to satisfy the demand for additional memory capacity because DRAM technology is hard to scale down to less than 20nm [7]. NAND flash has a high density and non-volatility while it does not support byte access and in-place update. These limitations of the traditional memory devices prompt researchers to intensively investigate the feasibility of innovative memory devices.

Several new non-volatile memory (NVM) technologies are emerged to address some of the shortcomings of the traditional memory devices. Phase change memory (PCM), spin-torque transfer random access memory (STT-RAM), and resistive random access memory (RRAM) offer high performance, byte-addressability, and large capacity. They consume low standby power due to its non-volatility. Therefore, they are expected to be a good candidate for DRAM replacement in the main memory system and flash replacement in the storage system [8, 9]. Moreover, the advantages of emerging NVMs even opens an option to unify the main memory and the storage system, and thus, many researchers investigate the feasibility of storage class memories [10, 11].

The undisputed benefits and huge potential of emerging NVMs justify the significant effort in researching the adoption of the emerging NVMs to the memory hierarchy by reinforcing the advantages of emerging NVMs and overcoming the shortcomings of them. However, we found that many parameters and assumptions from research papers in academia do not reflect recent technology trends in the industry, although their contributions are significant. Most designs concentrated on increasing system performance by reducing the number of memory accesses or optimizing the internal operations within emerging NVM devices, and this is due to the assumption that NVMs would use as similar architecture DRAM devices we using [12, 13]. This assumption is partially true, but they overlooked the time consuming interface operation in real industry NVM prototypes equipped

with low power double data rate 2 non-volatile memory (LPDDR2-NVM) [14, 15]. For NVMs, LPDDR2-NVM is the state-of-the-art standard of Joint Electron Device Engineering Council (JEDEC). In addition, most research assume too optimistic write operations, which is also far from the industry prototype. These non-realistic and optimistic assumptions in academia make it difficult to realize the many proposed ideas in real industry NVM prototypes.

1.2 Research Contributions

The main contribution of this dissertation is system-level performance optimization of memory system with LPDDR2-NVM by investigating the difference of conventional DRAM and LPDDR2-NVM. More specifically, this dissertation focuses on the way of interfacing with LPDDR2-NVM.

Contributions of this dissertation are summarized as follows.

- An LPDDR2-NVM prototype implementation in order to extract actual parameters of memory system using LPDDR2-NVM.
- A cycle-accurate system-level simulator implementation for evaluating the performance of memory interface technique for a LPDDR2-NVM based system.
- Design space exploration of the row buffer management policy and row buffer configuration in the LPDDR2-NVM for performance optimization.
- Propose a system-level performance optimization method for memory system with LPDDR2-NVM by utilizing the existing resources defined in the LPDDR2-NVM standard.

- Propose a proactive row buffer management method that mimics a reconfigurable row buffer architecture to optimize the performance of memory system with LPDDR2-NVM in system-level.

1.3 Organization of Dissertation

Chapter 2 discusses the background and related work on adopting emerging NVMs to the memory hierarchy. Chapter 3 introduces the LPDDR2-NVM platform to explore the relation between addressing architecture and system-level performance. Chapter 4 introduces experimental results and discusses the row buffer management policy and row buffer configuration. Chapter 5 introduces our system-level performance optimization effort for LPDDR2-NVM memory system. Chapter 6 concludes this dissertation.

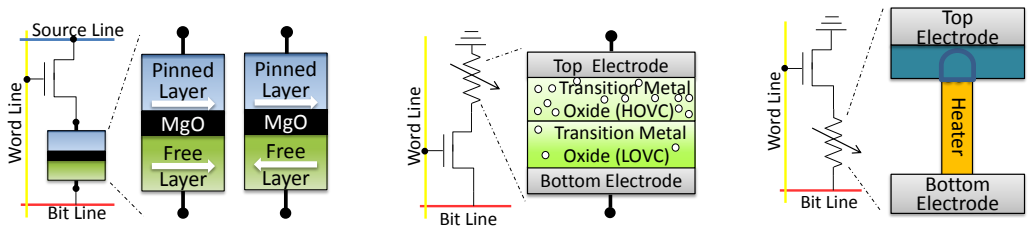
Chapter 2

Background

NVM is a resistive memory that uses a difference of the resistance to store data, and the characteristics of the NVM come from the storage elements. We present the basics of NVM in this chapter. We discuss the LPDDR2-NVM standard that is the state-of-the-art industrial standard for NVM. It has a different architecture and addressing mechanism to that of conventional DRAM. We focus on these differences. We also review the previous work for adopting the NVMs to memory hierarchies in this chapter.

2.1 Basics of Non-Volatile Memory

Emerging NVM technologies are based on the new type of storage elements. The STT-RAM uses a magnetic tunnel junction (MTJ) as a storage element, as shown in Figure 2.1(a). The orientation of magnetic layers determine the resistance of MTJ. A current of polarized electrons changes the orientation of free layer [16]. The RRAM consists of two metallic elements that sandwich a thin dielectric layer, as shown in Figure 2.1(b). An external voltage with specified polarity, magnitude, and duration changes the resistance of the RRAM [17].



(a) STT-RAM (b) RRAM (c) PCM

Figure 2.1: Cell structure of NVMs [5]

PCM uses a small volume of phase change material as a storage element, as shown in Figure 2.1(c). Read operation on the PCM measures the resistance of the cell by passing a current that is small enough not to change current state. It programs the cell by a proper heating and cooling [18].

The difference of storage elements of NVMs makes the different characteristics of NVMs. We compare the characteristics of emerging NVMs between the traditional memory device and NVMs, as shown in Table 2.1. The performance of DRAM is good while it consumes significant energy. DRAM is a volatile device, and it meets the scaling limitation. NAND has a high density due to its cell structure. However, it requires a complicate management scheme due to out-of-place update, limited write endurance, and poor write performance. STT-RAM has high performance and consumes low energy. Write endurance of STT-RAM is good, but it has relatively low density than other NVMs [16]. RRAM has high performance, high density, and low energy consumption, but it is still under the development. PCM has received considerable attention as a promising next generation NVM because of its scalability, fast byte access capability, low-power consumption and no requirement for erase-before-program [19]. However, write operation on PCM incurs high latency and high energy consumption, and write endurance is still not enough [14, 15].

Table 2.1: Comparison of memory device characteristics [1, 2, 3, 4].

Memory type	DRAM	NAND flash	STT-RAM	RRAM	PCM
Cell structure	1T1C	1T	1T1MTJ	1T1R	1T1R
Cell area	$6\sim8F^2$	$4\sim6F^2$	$6\sim20F^2$	$4\sim10F^2$	$4F^2$
Read time	$\sim$10ns	$20\sim100\mu$s	$1\sim10$ns	$5\sim10$ns	$10\sim40$ns
Write time	$\sim$10ns	$100\sim800\mu$s	$2\sim20$ns	$10\sim20$ns	$50\sim120$ns
Read energy	medium	low	low	low	low
Write energy	medium	high	low	low	high
Write endurance	10^{15}	$10^3\sim10^5$	10^{16}	10^8	10^8

2.2 LPDDR2-NVM

LPDDR2-NVM has been deemed the standard interface to connect NVMs. PCM prototypes from several manufacturers have been announced with an LPDDR2-NVM interface [14, 15]. LPDDR2-NVM standard somewhat similar to conventional DRAM, but at the same time, has different features due to the differences between conventional DRAM and NVMs including asymmetric read and write operations. The representative features of LPDDR2-NVM compared to the conventional DDR interface are:

- No precharge operation because of the non-destructive operation of non-volatile memory devices.
- Three-phase addressing mechanism for supporting large size of memory (up to 32Gb).
- No multi-bank architecture.
- Multiple row address buffers (RABs) and row data buffers (RDBs) which are selected

by the memory controller regardless of the physically accessed address.

- Smaller unit size of RDB (typically 32 bytes) than that of the conventional DRAMs.
- Indirect write operations via overlay window.
- Multiple partition architecture.
- Dual operation that enables read in other partition during cell programming.

We study the details of LPDDR2-NVM standard including architecture and the overlay window operations, and discuss the difference between conventional DRAM and LPDDR2-NVM in the following subsections.

2.2.1 Architecture

Figure 2.2 shows the functional block diagram of LPDDR2-NVM standard-compatible memory device. In LPDDR2-NVM standard, address and commands are transferred through command/address (CA) pins while conventional DRAM has dedicated 12 to 16 pins for transferring the address and command separately. LPDDR2-NVM specifies 10 bits of CA pins and they are used with DDR architecture even for the address phase. This indicates that the memory controller transfers up to 20 bits of command and address bits together per a memory clock cycle.

LPDDR2-NVM standard requires a longer row address to support large memory sizes of up to 32Gb. The longer row address cannot be transferred in a single operation due to the limited number of CA pins. Therefore, LPDDR2-NVM device has RABs that store the upper part of row address.

The cell programming in NVMs takes longer time than read usually. This asymmetric read and write operation led to use different mechanism for reading and programming mem-

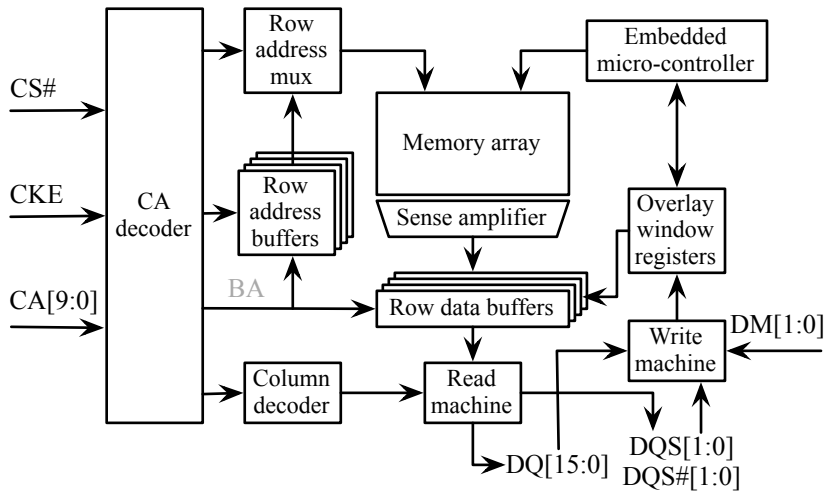


Figure 2.2: Functional block diagram of an LPDDR2-NVM compatible memory device.

ory array. LPDDR2-NVM standard-compatible device has an embedded micro-controller and overlay window registers to alleviate the problems cause by the long cell programming time, as shown in Figure 2.2.

2.2.2 Operation of overlay window

The process of read operation of LPDDR2-NVM is very similar to conventional DRAM. However, write operation – strictly speaking non-volatile cell programming – is completely different from conventional DRAM. Write operation is done indirectly through the special registers called overlay window similar to the method used for accessing NOR flash, as shown in Figure 2.3.

The overlay window consists of memory-mapped registers to control an LPDDR2-NVM device. It contains the command address register, the command code register, the command execution register, the program buffer, and so on. The size of overlay window is 4KB, and the location overlaps the address space of an LPDDR2-NVM device, as shown in

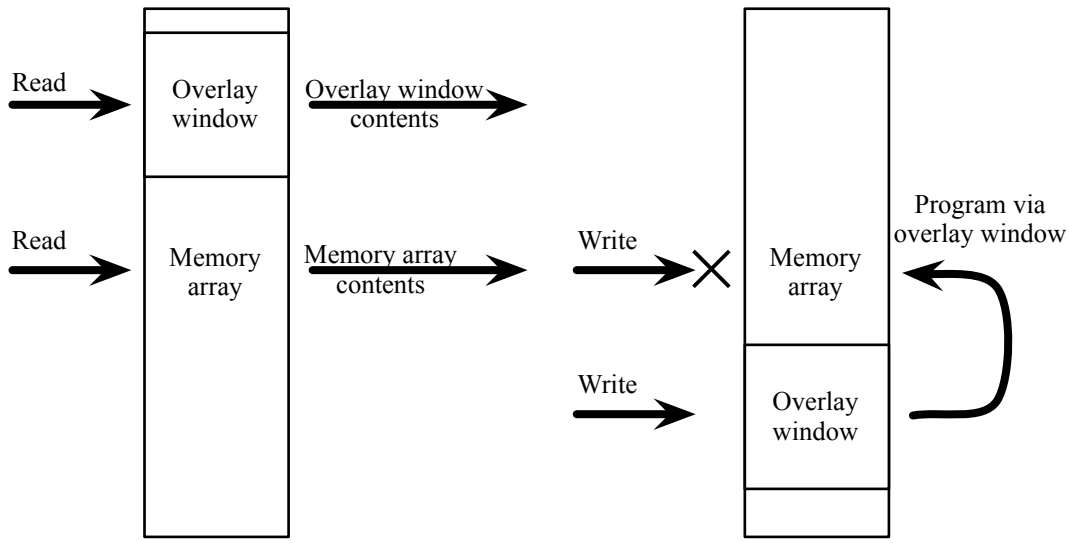


Figure 2.3: Comparison of read and write operation.

Figure 2.3. Mode registers enable or disable the overlay window, and set its location. Read operation to the address space overlapped by the overlay window accesses the contents of overlay window registers when the overlay window is enabled.

An LPDDR2-NVM standard supports several commands which make use of the overlay window, such as single word overwrite, buffered overwrite, suspend, and so on. A write operation should be translated into a sequence of overlay window accesses, as shown in Figure 2.4. Therefore, it incurs significant time overhead to interface with LPDDR2-NVM device, whereas a write operation in conventional DRAM uses the same interface of read operation.

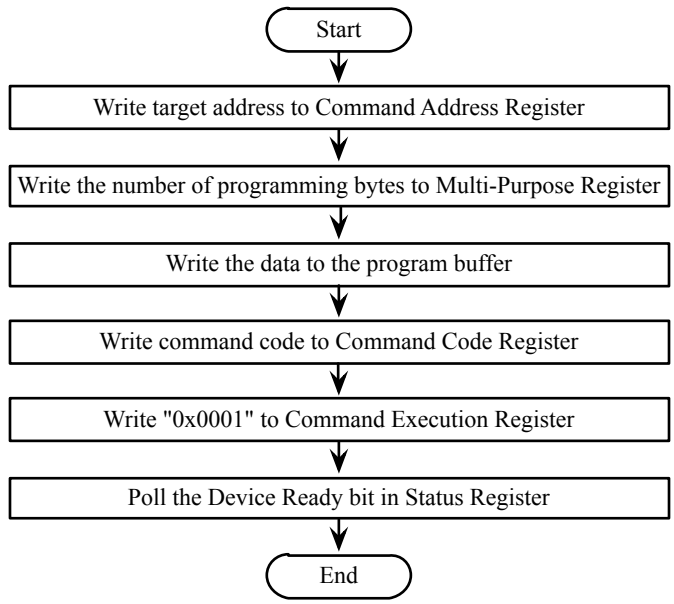


Figure 2.4: Flowchart of buffered overwrite [6].

2.2.3 Comparison to conventional DRAM

2.2.3.1 Row buffer architecture

LPDDR2-NVM has RABs to store the part of row address, and which do not exist in conventional DRAM. It also has RDBs that store a data like a row buffer in conventional DRAM. The RABs and RDBs are used as a pair in addressing mechanism. The BA signals are used to select a row buffer pair in LPDDR2-NVM while it is used to select a bank in the conventional DRAM. Figure 2.5(a) shows the address components of the conventional DRAM which has 8 banks. The address consists of bank address, row address, and column address, so the BA signals are used to select bank according to the physically accessed array address. On the other hand, the address of LPDDR2-NVM does not contain BA signals, as shown in Figure 2.5. BA signals are only intended to select a row buffer pair not a physical bank address of the memory array. The memory controller selects a proper

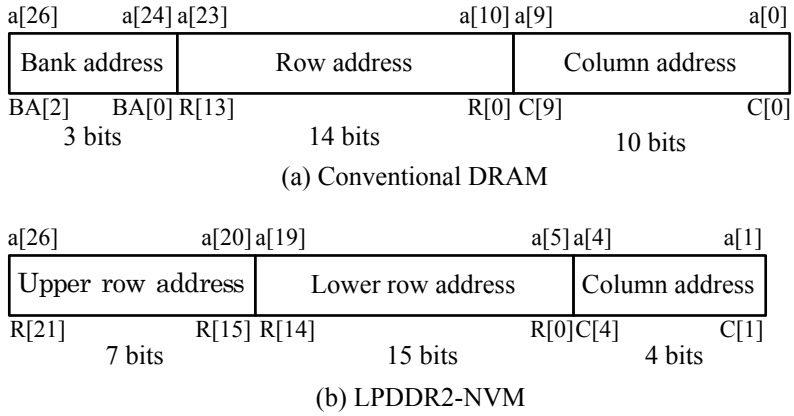


Figure 2.5: Comparison of address composition.

RAB and/or RDB by controlling these BA signals regardless of the physically accessed memory address. The least significant column address, C0, is implied to be zero and is not transmitted [6].

The conventional DRAM should close an open row within a timing constraint, t_{RAS}, using a precharge command. It also should close the open row before accessing other row in the same bank. The precharge operation invalidates the data in the row buffer. However, LPDDR2-NVM does not need a precharge command because it uses a current sense amplifier instead of voltage sense amplifier. The data in RDB is valid until the power is off, so the timing constraint between activate and read/write, t_{RCD}, has only minimum value. However, the data in RDB is not updated automatically when the contents of memory array changes. It means that incoherency problem between the RDB and memory array is able to happen. The memory controller should track the validity of RDBs to avoid this incoherency problem.

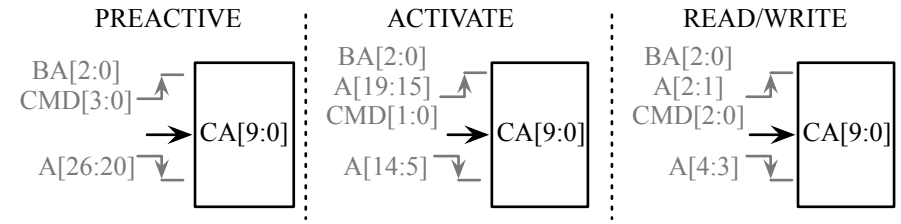
In multi-bank architecture of conventional DRAM, rows in other banks can be activated at the same time as long as it meets the timing constraints such as the minimum time interval between Activate commands to different banks, t_{RRD}, and four bank activate window, t_{FAW}.

The LPDDR2-NVM supports multiple (4 or 8) row buffers instead of multiple banks. The timing constraints such as t_{RRD} are applied to bank operations are applied to row buffer operations in the LPDDR2-NVM. The memory controller is able to activate one row buffer while the other row is activating if the timing constraints are satisfied.

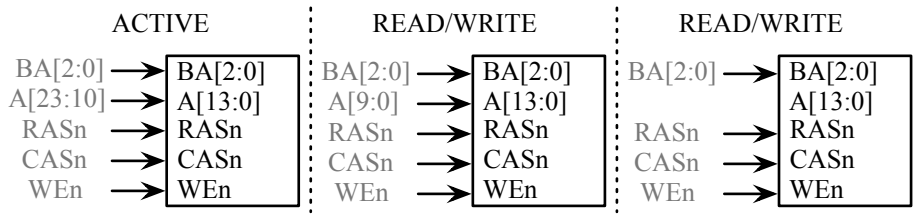
2.2.3.2 Addressing mechanism

In LPDDR2-NVM, three-phase addressing mechanism is used to support larger size of memory devices as opposed to conventional DRAM using two-phase address mechanism. Three-phase addressing consists of preactive, activate, and read/write phases. Figure 2.6 shows the comparison of used pins and transferred information at each address phase. LPDDR-NVM and conventional DRAM have same density, 1 Gb, and data width, 8 bits. As described in the Section 2.2.1, CA pins use DDR architecture in LPDDR2-NVM. In preactive phase, command information and BA signals used to select RAB transferred at the rising edge of the clock, and upper row address is transferred at the falling edge of the clock. Command information, BA signals to select RAB and RDB, and part of lower row address are transferred at the rising edge of the clock in activate phase. At the falling edge of the clock in activate phase, the remaining lower row address are transferred. In read/write phase, command information, BA signals to select RDB, and part of column address are transferred at the rising edge of the clock, and then the remaining column address is transferred at the falling edge of the clock.

Figure 2.7 shows the detailed behavior of three-phase read operation. In preactive phase, only upper 3 to 12 bits of the row address are transferred and this partial row address is stored into the designated RAB. The BA signals are used to select a designated RAB. In activate phase, remaining row address is transferred. The entire row address after combining it with the upper row address stored in the RAB is select row of the memory array,



(a) LPDDR2-NVM addressing



(b) Conventional DRAM addressing

Figure 2.6: Comparison of address and command pins and transferred information.

and then the corresponding row data is transferred from the memory array to the designated RDB. The RAB and RDB pair is selected by BA signals. The data is transferred from the RDB selected by BA signals to the memory controller at the last phase. The size of upper row bits, lower row bits, and column bits are determined by the density of device and the unit size of RDB.

It is possible to operate several read and/or write access at the same time if it uses different RAB and RDB pairs and satisfy timing constraints. It is similar to interleaving operation of conventional DRAM. Like an open-page policy in conventional DRAM, the preactive phase or activate phase of three-phase addressing can be omitted if the RAB or RDB has valid data.

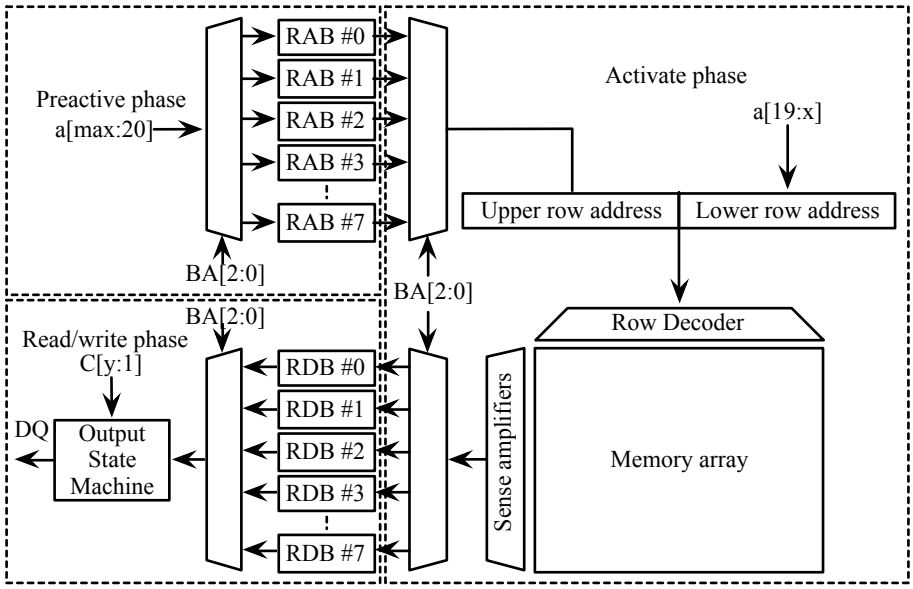


Figure 2.7: Operation of three-phase read operation [6].

2.2.3.3 Dual operation

The NVMs usually has a low write performance due to a long cell program time. This long cell program time also degrades read performance when a write access blocks read access. Multiple partition architecture and dual operation are introduced to alleviate this read performance degradation in LPDDR2-NVM.

Dual operation allows read operation in other partition while programming or erasing in one partition. The read operation should be postponed if it tries to access the partition that is programming cells. This dual operation increases the performance of LPDDR2-NVM although programing in other partition is not allowed during programming or erasing in one partition. This is a distinctive feature of LPDDR2-NVM because conventional DRAM does not allow read operation during write operation.

2.3 Related Work

In this section, we review the related work on adopting NVMs to the memory hierarchies. We discuss what the weakness of the traditional memory device is and how NVMs cover that by focusing on the architectural perspectives of the previous work.

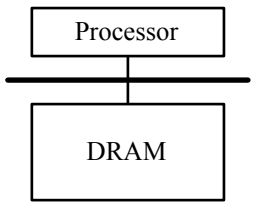
Cache There is a performance gap between a processor and a main memory. Usually, cache is used to reduce this performance gap, so cache requires high performance. SRAM is main component of the cache thanks to its high performance. Cache size affects system performance significantly because larger cache can hold more data and reduce cache misses. The size of SRAM cache is limited due to large footprint. SRAM cache consumes large leakage power because of non-volatility of SRAM. Therefore, system performance increases and energy consumption is reduced if the new memory device which has high performance, small footprint, and non-volatility is used in the cache instead of SRAM.

Recently, several NVM technologies such as PCM, STT-RAM, and RRAM are considered as alternative of SRAM in the cache, but these NVMs also do not meet the all of the ideal cache requirements. STT-RAM has a high read speed, but a relatively large footprint [16]. PCM has small footprint comparable to DRAM and less leakage power consumption thanks to its non-volatility [20]. High write latency, write energy and limited write endurance incurs a problem when we use PCM as a cache. Slower read performance of PCM than that of SRAM degrades system performance. However, its small footprint and, in turn, large capacity compensates this performance degradation in some cases.

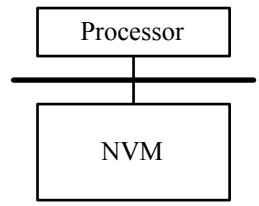
Cache memory requires high performance, but long write latency of STT-RAM prevents wide adoption in cache. However, the write latency of STT-RAM can be reduced by relaxing the data retention time of STT-RAM. This volatile STT-RAM requires refresh op-

eration to prevent data loss, but it still reduces the energy consumption [21, 22, 23]. Some researches have been dedicated to solve the problem of PCM as a cache memory. The hybrid cache architecture that consists of small SRAM and large PCM has been proposed to reduce leakage energy in L1 instruction cache and L2 unified cache [24]. A trade-off between performance and power was reported in the several hybrid non-uniform cache architecture (NUCA) under the same area constraints. These hybrid cache architecture use cache line allocation and migration policy that considers the different characteristics of SRAM and PCM. High density and low leakage power consumption of PCM enables additional L4 cache layer with negligible overhead [25]. Write reduction and distribution technique has been proposed to enhance energy reduction and prolong lifetime of pure L2 cache [20].

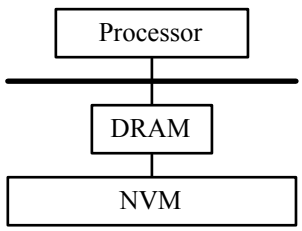
Main Memory Traditionally, main memory consists of DRAM, as shown in Figure 2.8(a). DRAM consumes significant amount of standby power due to its 1T1C cell structure and non-volatility. DRAM also meets the limitation in scaling as explained in Chapter 1. NVM, especially PCM, is considered as alternative of DRAM which is shown in Figure 2.8(b) because it has high density, byte-accessibility, low static power, and non-volatility [26, 13, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Write latency and energy consumption of PCM is higher than DRAM because it requires higher current and longer time to change its phase. This high write latency increases an effective read latency when read operation is blocked by write operation. It degrades system performance significantly. A limited endurance also prevents adopting PCM as a main memory. Hybrid architectures with DRAM have been proposed to alleviate performance degradation and short lifetime of PCM. Architecture uses small size of DRAM as cache to hide write latency and reduce write activity, as shown in Figure 2.8(c) [12, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55].



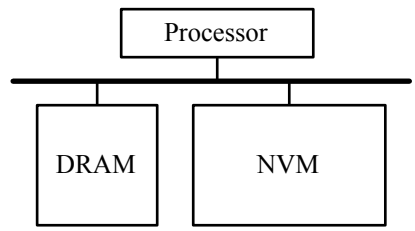
(a) Traditional main memory



(b) NVM-only main memory



(c) NVM main memory with a DRAM cache



(d) Hybrid main memory with DRAM and NVM

Figure 2.8: Comparison of the main memory architecture.

In the other hybrid architecture, DRAM is in the same level as NVM, as shown in Figure 2.8(d) [56, 57, 58, 59, 60, 61].

NVM-only main memory is introduced to reduce energy consumption by utilizing its zero leakage power in memory cell [26]. NVM-only main memory can reduce the access latency due to page fault because it has more capacity to hold most of the pages those are needs during program execution. However, the endurance of PCM is not enough for the main memory. It requires a technique to reduce the number of write to utilize the advantage of NVM-only main memory. First of all, system performance increases and energy consumption is reduced if the bit change decreases.

PCM is slower than DRAM especially in write, and it increases memory access latency. It also has limited write cycles compares to DRAM. Therefore, DRAM used as a buffer for PCM main memory to reduce a write operation [12]. Some approaches have been proposed to reduce PCM write through the DRAM cache. First, multiple dirty bits technique is pro-

posed. Multiple dirty bits keep the changes of a divided cacheline. It helps reducing write operation when a line is evicted from DRAM cache by writing only changed parts back to PCM main memory [12, 45, 46]. Second, PCM write is avoided by an additional status bit, present bit, in the DRAM cache [12]. Unlike usual operation, the data is only written to DRAM when missed page fetched from the storage. Present bit indicates the existence of page in the PCM. The data is only written to the PCM when present bit is clear or dirty bit is set at the eviction. Last, DRAM cache replacement algorithm reduces PCM write by considering the characteristics of PCM. Cache uses a least recently used (LRU) policy to increase hit ratio usually. It does not consider the asymmetry between read and write operation of PCM. N-chance algorithm prefers clean victim when it select a victim in the DRAM cache [45]. It first selects the oldest clean line among the N least recently used lines. It uses a LRU policy if such a line does not exist.

Another way to address the issues of PCM is a heterogeneous main memory architecture that consists of DRAM and PCM. It is important to utilize the different characteristics of DRAM and PCM. Performance enhancement, lifetime extension, and energy reduction can be archived by a proper data allocation and migration. DRAM is a write friendly device but incurs high energy consumption while PCM is a write hostile device. Write operation can be reduced if DRAM contains a frequently updated data, hot-data, and PCM contains an occasionally updated data, cold-data. This hot-cold separation can be done when data is evicted from cache, data allocation, or when write count on the segment exceeds threshold, data migration. The hot-cold separation mechanism can be implemented in the controller level [59] or by modifying OS virtual memory management scheme [57, 56, 58, 60]. The access pattern is monitored in the controller [57, 59] or the page table [58, 60]. In some case, the individual page can be migrated continuously. This Ping-Pong migration can be resolved by the adaptive page grouping (APG) based on the physical page frame number.

This hybrid main memory architecture is also used in application-specific DSP system. In the DSP system, variable partitioning and instruction scheduling problem is important because it determines system performance and energy consumption. These objectives, which are in the trade-off relation, can be efficiently tacked in the compilation time by jointly considering the power consumption and the number of write on PCM [61].

Storage System Recently, NAND flash is widely used as storage device thanks to its high density and lower read latency than a hard-disk drive. The NAND flash requires out-of-place update because erase operation must be done before program operation. The unit of erase operation is a block while the unit of read and program operation is a page. This mismatch requires a complicate mapping table to track the location of pages and, in turn, flash translation layer (FTL). A part of mapping table, metadata, keeps in the main memory during the system operation due to frequent updates. The remaining of the mapping table is stored in the NAND flash because the main memory is an expensive resource [62, 63].

The conventional storage architecture of the embedded system consists of DRAM, NOR flash, and NAND flash [64]. NOR flash is used as code storage because it supports execute-in-place (XIP). Metadata and user data are stored in the NAND flash. The density of PCM is close to NAND while it supports random access and in-place update [65]. The hybrid storage architecture with PCM and NAND has been proposed. It uses a PCM as metadata storage to improve the system performance and lifetime of NAND flash [64, 66, 67, 62, 68, 69]. The hybrid storage separates metadata of the filesystem and FTL metadata such as page mapping table, physical NAND block information, bad block management information, and so on. Similar approach that uses PCM as metadata storage in the solid-state disk (SSD) has been proposed [65, 63]. Filesystem that considers the hybrid architecture of storage such as PFFS and PFFS2 also has been proposed [66, 67, 62].

The metadata is stored in the DRAM in conventional storage architecture. It should be moved to NAND flash when a sudden power failure occurs. Otherwise, the system loses important mapping data and, in turn, the reliability of the system decreases [67, 70]. Moving data from DRAM to NAND flash requires a significant energy due to a power hungry NAND operation. The system reliability increases if the page mapping table is stored in the PCM [67] or a part of PCM is reserved for a sudden power failure [70]. In those cases, moving data from DRAM to NAND is not required, so the design of power failure protection mechanism can be simple [67].

High density of PCM makes it as NAND flash replacement in the storage. However, this is not widely accepted concept because PCM main memory is possible due to its byte-accessibility. MLC capability of PCM helps the adoption in storage system while MLC capability comes with a performance and a lifetime penalty. Run-time MLC/SLC reconfiguration mechanism has been proposed to compensate these penalties. PCM based SSD that connects with the host via PCIe also has been proposed [71].

Chapter 3

Implementation of LPDDR2-NVM Platform

The previous chapter has shown that a LPDDR2-NVM has a different architecture and operation from a conventional DRAM. Therefore, LPDDR2-NVM platform is necessary to evaluate and optimize the performance of memory system that uses LPDDR2-NVM as a main memory.

This chapter focuses on implementation of LPDDR2-NVM platform. The LPDDR2-NVM platform is made up of a LPDDR2-NVM prototype and a system-level simulator. LPDDR2-NVM prototype verifies the operation of memory system with LPDDR2-NVM, and it is used to extract parameters of memory system with LPDDR2-NVM. We implement a system-level simulator with the extracted parameters because of LPDDR2-NVM prototype's limited capability due to lack of flexibility.

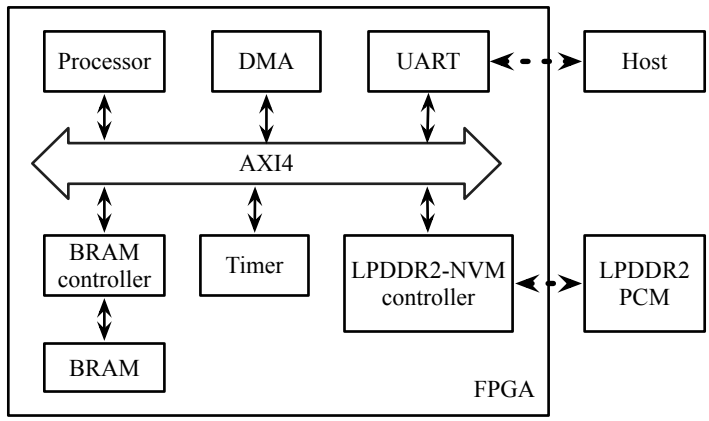


Figure 3.1: Block diagram of an LPDDR2-NVM control SoC.

3.1 LPDDR2-NVM Prototype

LPDDR2-NVM prototype consists of a control system-on-chip (SoC), an LPDDR2-NVM small outline dual in-line memory modules (SODIMMs), and a FPGA board. We use a PCM device with LPDDR2-NVM interface to implement a real LPDDR2-NVM memory system.

The control SoC includes a MicroBlaze processor, our customized LPDDR2-NVM controller, and many other conventional components, as shown in Figure 3.1. It uses Advance eXtensible Interface 4 (AXI4) as a system bus. All components are implemented in a field-programmable gate array (FPGA). Application and overlay window management code operate at the MicroBlaze processor. The customized LPDDR2-NVM controller interfaces with a LPDDR2-NVM SODIMM. The timer measures memory access time to evaluate the performance of memory system. The processor operates at 100 MHz, and the LPDDR2-NVM SODIMM operates at LPDDR2-400.

We implemented an FPGA board with a Xilinx XC7K325T-2FFG900C that contains two DDR3 SODIMM sockets, as shown in Figure 3.2. This FPGA board enable to evaluate

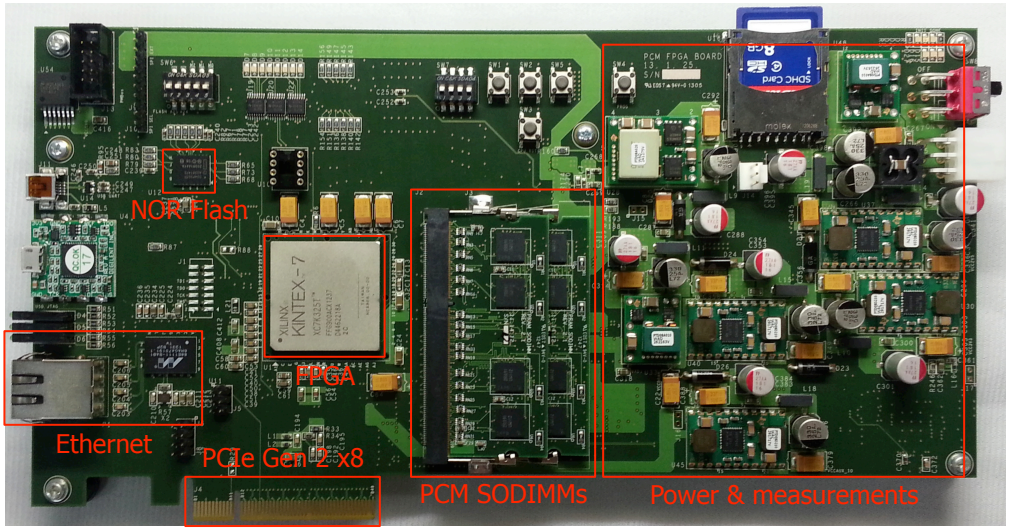


Figure 3.2: FPGA board for an LPDDR2-NVM prototype.

the performance of LPDDR2-NVM when it uses for main memory system and storage system because it has a 128MB NOR flash and supports PCIe Gen2 8-lane and10/100/1000 tri-speed ethernet. The FPGA board supports power consumption measurement of power domains.

We develop a console program that supports memory dump, memory copy, and programming PCM. It operates at the control SoC and verifies the operation of prototype. The operation of this prototype is also verified by using conventional memory test routines such as walking-0, walking-1, incremental address, inverse address, and fixed patterns.

3.1.1 LPDDR2-NVM controller

Figure 3.3 shows the block diagram of the customized LPDDR2-NVM memory controller. The controller consists of an AXI4 interface, several state machines to guarantee the timing constraints, and the physical layer (PHY) to support DDR architecture. The state machine of the LPDDR2-NVM controller is different from that of conventional DRAM controllers

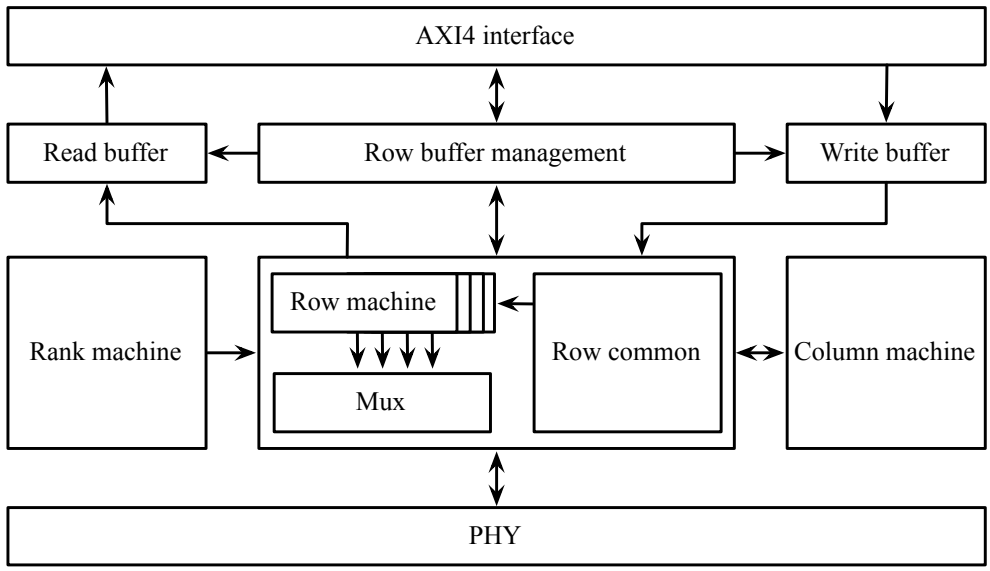
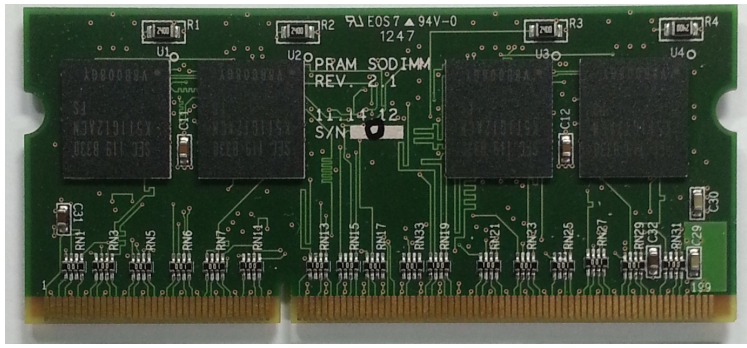


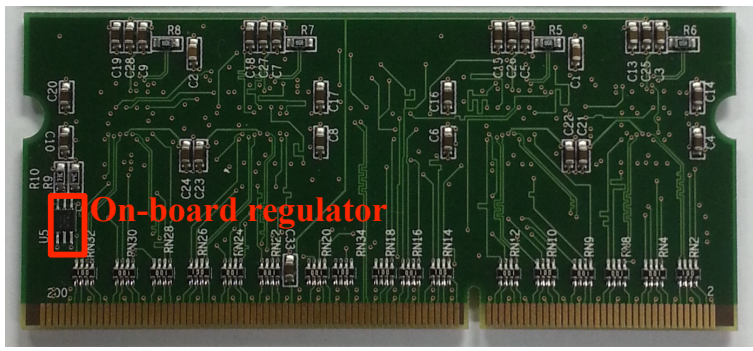
Figure 3.3: Block diagram of an LPDDR2-NVM controller.

because the LPDDR2-NVM uses three-phase addressing and does not require precharge and refresh operations.

The row buffer management module controls the command flow, decides start address phase and row buffer pair, and manages validity of row buffer pair. The rank machine module manages t_{RRD} and write to read delay constraint. The row machine manages the memory access from the addressing phase decided by the row buffer management module. It manages t_{RCD}, t_{RAS}, and t_{RP}. The customized LPDDR2-NVM memory controller has several row machines, as shown in Figure 3.3. These several row machines enable that the memory controller manages several accesses simultaneously. DQ bus and data transfer is controlled by the column machine. PHY module converts data rate to interface with LPDDR2-NVM and initiates the LPDDR2-NVM devices after the power-up sequence.



(a) Frontside



(b) Backside

Figure 3.4: LPDDR2-NVM SODIMM board.

3.1.2 LPDDR2-NVM SODIMM

NVMs have limited write endurance, so we have to replace NVMs if it wears out. It is a reasonable approach to use a SODIMM instead of discrete components because it enable us to replace the wear-out NVMs easily without soldering. However, there is no JEDEC standard for an LPDDR2-NVM SODIMM, so we adopt a JEDEC DDR3 SODIMM standard to implement an LPDDR2-NVM SODIMM, as shown in Figure 3.4. An LPDDR2-NVM SODIMM contains four industry prototype PCM chips with an LPDDR2-NVM interface. The total capacity of an LPDDR2-NVM SODIMM is 512MB and the data width is 64 bits. Our SODIMM operates at LPDDR2-400.

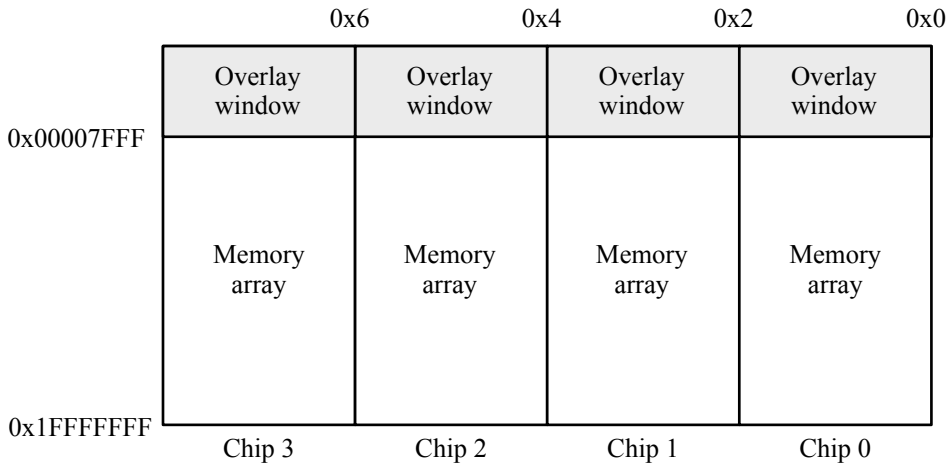


Figure 3.5: Address space of an LPDDR2-NVM SODIMM.

The prototype of PCM with LPDDR2-NVM interface uses 1.8V and 1.2V as a core power supply and 1.2V as an input buffer and I/O buffer power supply. It also uses 0.6V as a reference voltage. 1.2V and 0.6V are supplied to LPDDR2-NVM SODIMM through pins of SODIMM, but 1.8V is supplied by on-board regulator located on the backside of SODIMM, as shown in Figure 3.4(b). LPDDR2-NVM SODIMM operates at high speed, so it should be designed with consideration for the signal integrity. We applied line length rule for clock signals, DQ signals group, and control signal group. We swap the bits in the data byte group to reduce routing complexity.

The LPDDR2-NVM SODIMM widens the data width by placing PCM chips in parallel, as shown in Figure 3.5. This incurs an address mapping problem for write operation through overlay window because the overlay window uses a different address space from the address space of the processor. We implement an address mapping scheme for overlay window accesses.

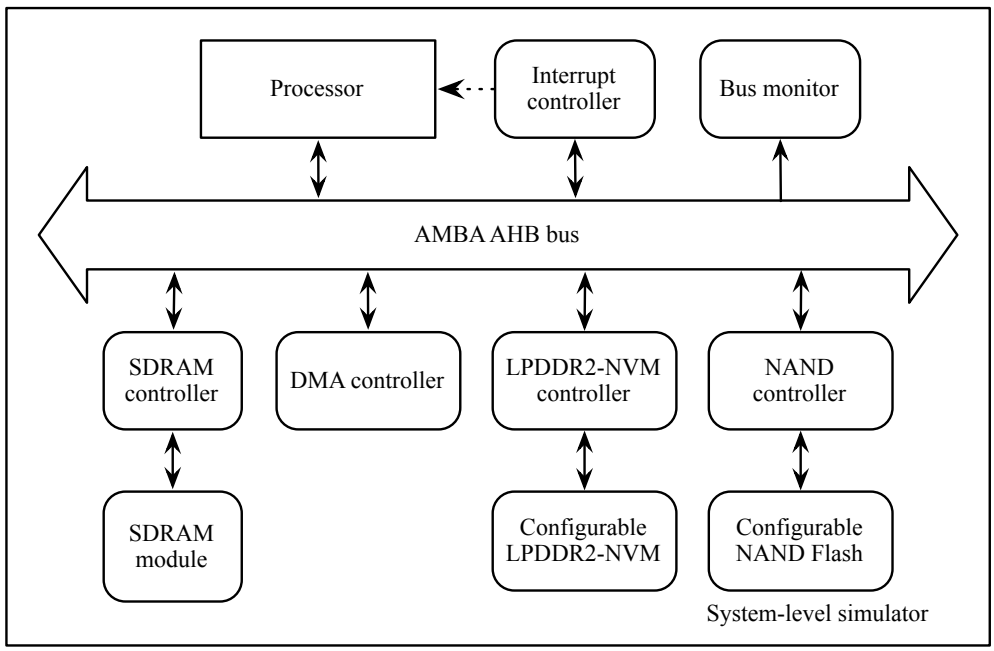


Figure 3.6: Block diagram of a system-level LPDDR2-NVM simulator.

3.2 System-Level Simulator

3.2.1 Architecture

We develop a cycle-accurate trace-driven SystemC simulator that evaluates the memory access time including bus transaction time. Figure 3.6 shows a block diagram of a system-level LPDDR2-NVM simulator. It uses an AMBA AHB bus as a system bus that supports multiple master environments. The component parameters such as clock period of system bus and IPs, latency of memories, and initial data of memories are configurable through the parameter files.

The processor module emulates the operation of processor using the trace from other simulator. The bus monitor module logs details of bus transaction. DMA module supports fast data move, and we can simulate a multiple master situation using a DMA module.

Table 3.1: Configuration of the emulated processor.

System	8 cores in-order processor
Processor	UltraSPARC-III+, 2GHz
L1 cache (Private)	I- and D-cache: 64KB, 4-way 64B block
L2 cache (Shared)	1MB, 4-way 64B block
Main memory	1GB

LPDDR2-NVM controller module emulates the behavior of LPDDR2-NVM controller in the LPDDR2-NVM prototype, and LPDDR2-NVM module emulates the behavior of LPDDR2-NVM SODIMM.

3.2.2 Processor modeling

The processor module uses a trace to emulate an 8-core in-order processor system that operates at 2GHz clock frequency. Table 3.1 shows the details of the emulated processor configurations. The processor module generates a burst AHB transaction using the memory access information from the trace. It uses the processor clock cycle information to generate idle cycles between burst transactions. Idle bus cycles is calculated from the clock period of bus and processor. The processor module passes the calculated cycles after it completes the previous memory transaction.

The traces have been extracted from the Simics full-system simulator [72]. For trace extraction, we select thirteen multi-threaded benchmarks from the PARSEC benchmark suite [73]. Table 3.2 summarizes the characteristics of the each benchmark in terms of the ratio of read operations normalized to the write operations and the frequency of the memory accesses.

Table 3.2: Memory access characteristics of the benchmark applications.

Application	R/W ratio	Memory accesses / 1k CPU cycles
blackscholes	5.96	0.23
bodytrack	5.96	0.83
canneal	1.84	12.62
dedup	1.39	4.74
facesim	6.58	0.22
ferret	1.52	0.91
fluidanimate	1.39	4.74
freqmine	1.55	1.63
raytrace	27.54	0.27
streamcluster	8.24	0.15
swaptions	11.11	0.58
vips	1.32	1.64
x264	1.78	2.32

3.2.3 LPDDR2-NVM modeling

The simulator configures the parameters of LPDDR2-NVM controller and memory devices such as operation speed, density, and timing parameters a through configuration file. We assume that the emulated system uses an LPDDR2-NVM SODIMM that contains 4 LPDDR2-NVM chips. The data width of the LPDDR2-NVM SODIMM is 64 bits, and 4 chips in the LPDDR2-NVM SODIMM operates simultaneously. The cell program time of a commercial PCM device with an LPDDR2-NVM interface is 20μs [14], while the very optimistic cell program time presented in the research prototype is 150ns [15]. Our LPDDR2-NVM model considers this varying cell program time. It supports multiple parti-

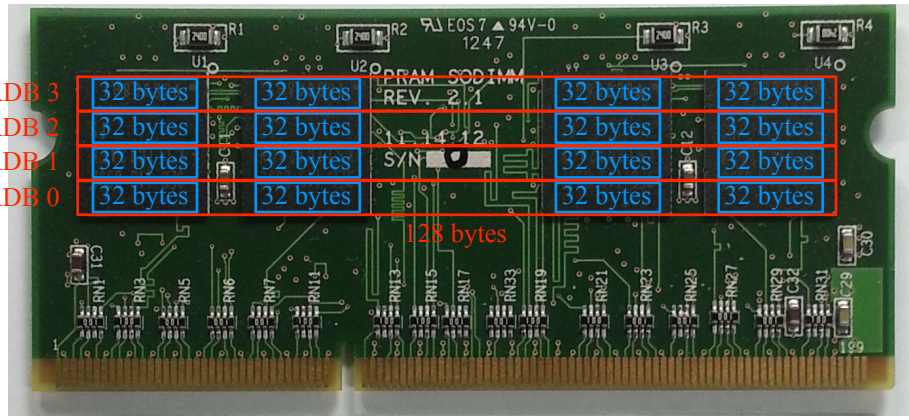


Figure 3.7: RDB configuration in an LPDDR2-NVM SODIMM.

tion architecture and dual operation.

The performance of memory system with LPDDR2-NVM varies on the RDB configuration, the unit size of RDB and the number of RDBs. We use the notation like [the number of RDBs]×[unit size of RDB] to indicate RDB configuration. Note that the unit size of RDB implies the unit size of RDB in a LPDDR2-NVM SODIMM. For example, figure 3.7 shows the 4×128 bytes RDB configurations. Each LPDDR2-NVM chip has four 32 bytes RDBs, so the unit size of RDB is 128 bytes.

The most important thing in a cycle-accurate modeling is managing timing constraints correctly. The LPDDR2-NVM model manages timing constraints using several counters. The LPDDR2-NVM model manages several memory accesses simultaneously to emulate the prototype accurately. Specific data structures are also used to track the status of RDBs and DQ.

Chapter 4

Design Space Exploration of Row Buffer Architecture in LPDDR2-NVM

This chapter discusses the effect of row buffer management policy and configuration on the performance of memory system with LPDDR2-NVM. The RDB hit ratio affects the performance of memory system, and the RDB hit ratio varies according to the row buffer management policy and configuration.

We evaluate the performance using the system-level simulator with the benchmark described in section 3.2. The parameters used in the evaluation are summarized in Table 4.1.

4.1 Row Buffer Management Policy

As described in Section 2.2, the memory controller selects a row buffer in the LPDDR2-NVM like a fully-associative cache while the row buffer selection in conventional DRAM

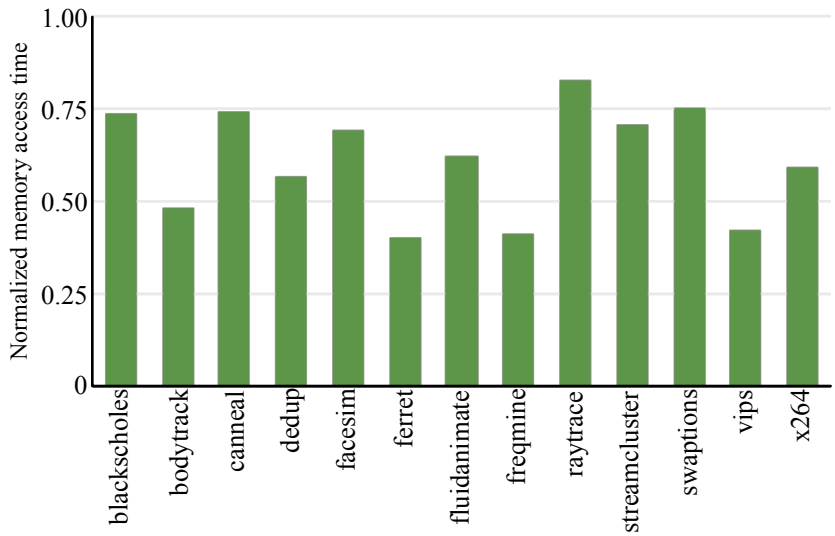
Table 4.1: Configuration of the simulated LPDDR2-NVM.

LPDDR2-NVM main memory	1GB, LPDDR2-800, 64-bit wide
Preactive to Activate (t_{RP})	3 t_{CK}[1]
Activate to Read/Write (t_{RCD})	120ns
Read latency (RL)	6 t_{CK}[1]
Write latency (WL)	3 t_{CK}[1]
Burst length (BL)	8
Cell program time ($t_{program}$)	150ns
The number of partitions	16

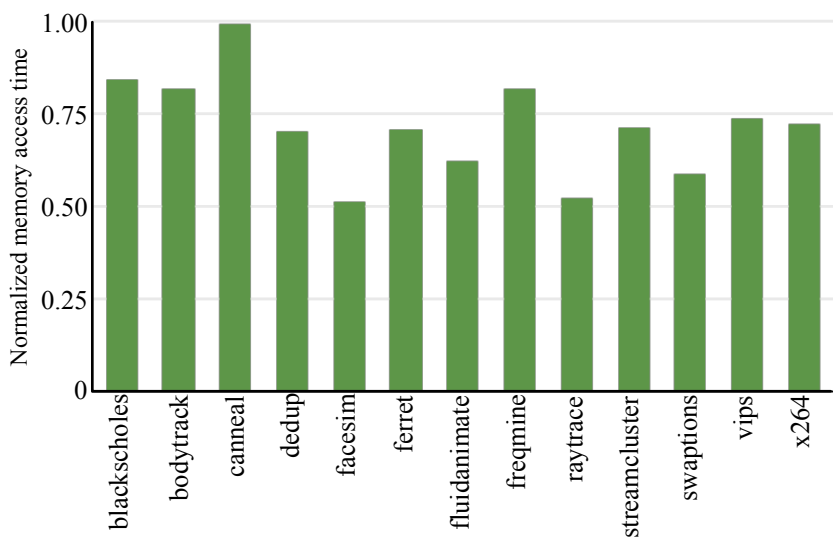
[1] t_{CK} is a memory clock cycle (2.5ns at LPDDR2-800)

is fixed by the internal architecture of DRAM similar to a directed-mapped cache. This architecture in the LPDDR2-NVM enables us to choose a proper row buffer management policy. The difference of management policy causes a different RDB hit ratio during read and write operations and this, in turn, leads to different memory access time.

Figure 4.1 shows the comparison of the memory access time between a direct-mapped policy and a fully-associative mapping policy. We evaluate two RDB configurations: 4×128 bytes RDBs configuration and $8\times16{,}384$ bytes RDBs configuration, respectively. The memory access time of the fully-associative mapping policy is normalized to that of the direct-mapped policy. The commercial LPDDR2-NVM device has 4×128 bytes RDBs configuration while the $8\times16{,}384$ bytes RDBs configuration is the largest capable RDB configuration in the LPDDR2-NVM standard. Table 4.2 shows the RDB hit ratio of read access, r_{RD}, and overlay window access, r_{OW}, when a direct-mapped policy or a fully associative mapping policy is used.



(a) 4×128 bytes RDBs



(b) 8×16,384 bytes RDBs

Figure 4.1: Memory access time of the fully-associative mapping policy normalized to the direct-mapped policy (lower is better).

Table 4.2: Comparison of the RDB hit ratio.

Application	4×128 bytes RDBs				8×16,384 bytes RDBs			
	Direct-mapped		Fully-associative		Direct-mapped		Fully-associative	
	r_{RD}	r_{OW}	r_{RD}	r_{OW}	r_{RD}	r_{OW}	r_{RD}	r_{OW}
blackscholes	9.2	25.0	15.1	70.2	22.8	99.2	41.8	99.6
bodytrack	30.0	25.0	28.7	64.4	74.5	99.2	87.2	99.9
canneal	0.2	25.0	0.2	47.0	1.9	99.9	4.3	99.9
dedup	19.2	25.0	13.8	60.6	49.3	99.7	88.3	99.9
facesim	28.3	25.0	36.7	59.1	62.0	82.9	92.9	99.6
ferret	35.8	25.0	38.8	76.5	78.1	96.1	95.1	99.9
fluidanimate	4.2	25.0	4.9	56.7	17.5	99.7	87.1	99.9
freqmine	35.1	25.0	35.0	72.3	74.3	99.6	88.2	99.9
raytrace	15.5	25.0	28.7	45.4	49.0	88.7	87.5	97.1
streamcluster	30.2	25.0	35.9	62.1	65.9	99.2	85.4	99.7
swaptions	37.2	25.0	44.5	61.1	76.5	75.8	95.0	99.8
vips	25.6	25.0	25.7	69.9	65.5	99.2	89.6	99.9
x264	16.6	25.0	18.5	58.5	57.5	98.8	84.7	99.9

Overall, the fully-associate mapping policy outperforms the direct-mapped policy in all applications and configurations. In LPDDR2-NVM devices, one write operation requires several consecutive accesses to the control registers located in the overlay window where the row address of control registers are closed each other. Therefore, in direct-mapped policy, only one RDB is allocated to the entire address space of the overlay window and this RDB is continuously selected as a victim. This happens even though the other RDBs are available like a conflict miss in a cache. On the other hand, the conflict miss is seldom happen in the fully-associative mapping policy because several RDBs are allocated

for the address space of overlay window. This indicates almost no conflict miss during a write operation and only a capacity miss is happened in a fully-associative mapping policy. Therefore, r_{OW} of direct-mapped policy is only 25.0% while r_{OW} of fully-associative policy is on average 61.8% in 4×128 bytes RDBs configuration.

The performance gaps between two management policies are reduced when the unit size of RDB and the number of RDBs are increased, as shown in Figure 4.1(b). One 16KB RDB can hold entire space of the overlay window even in the direct-mapped policy. r_{OW} of the direct-mapped policy in 8×16,384 bytes RDBs configuration is close to that of the fully-associative policy in all applications, as shown in Table 4.2. However, r_{RD} of the direct-mapped policy is lower than that of the fully-associative policy due to the conflict miss, and it makes the difference in the memory access time.

We do not explorer the victim selection policy of the row buffer management because we found very small variations though we changed the victim selection policy to round robin, least recently used (LRU), and other conventional ones. Therefore, LRU is used for the rest of this dissertation unless otherwise stated.

4.2 Row Buffer Configuration

4.2.1 Unit size of row data buffer

We evaluate the effect of unit size of RDB on the memory access time, as shown in Figure 4.2. The plots with different markers and lines represent the results from the different application traces. The unit size of each RDB varies from 128 to 16,384 bytes while the number of RDBs is fixed to 4 for all configurations. We measure the memory access time using latency at the processor. We normalize the memory access time of each configuration to the baseline configuration (4×128 bytes RDBs).

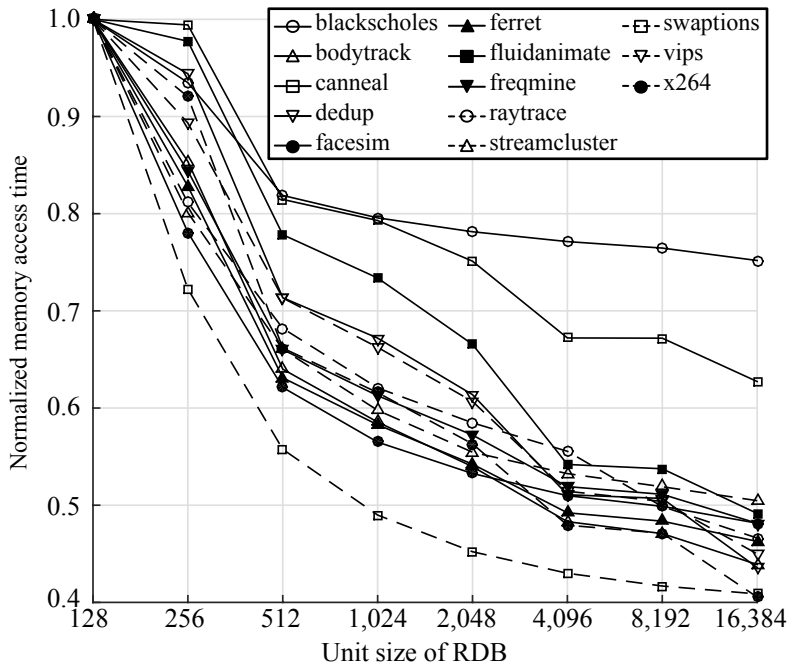


Figure 4.2: Memory access time varying on a unit size of RDB. The number of RDBs is set to four for all configurations. The values of each memory access time are normalized to the memory access time of 4×128 bytes RDBs configuration.

The comparison result shows that the effect of unit size of RDB varies on the behaviors of memory accesses. For instance, *blackscholes* – open circle with solid line – is almost not affected by the unit size of RDB. The reason is that increasing the unit size of RDB does not increase the hit ratio in RDBs significantly for *blackscholes* because the access pattern of the *blackscholes* is random. On the other hand, the memory access time of *swaptions* – open square with dash line – is reduced significantly as the unit size of RDB increases because the access pattern of *swaptions* is sequential.

Another observation is that memory time reduction by increasing the unit size of RDB is limited after the 2,048 bytes in most cases. Even in the most sensitive application, *swap-*

tions, the memory access time decreases just 4.3% when we change the unit size of RDB from 2,048 bytes to 16,384 bytes. The memory access time decreases significantly when the unit size of RDB changes from 8,192 to 16,384 bytes in some applications. We analyze that accessing overlay window causes frequent RDB misses in those applications when the unit size of RDB is less than 16,384 bytes. However, the number of misses is reduced when the unit size of RDB is the same as the size of overlay window. The experimental results imply that there are a lot of possibilities to optimize the row buffer space if we adequately consider the behaviors of the memory accesses.

4.2.2 Number of row data buffers

We perform another design space exploration by changing the number of RDBs, as shown in Figure 4.3. Instead of increasing the unit size of RDB, we change the number of RDBs from 2 to 128. In this analysis, the unit size of RDB is fixed to 128 bytes for all configurations. We normalize the memory access time of each configuration to the baseline configuration (4×128 bytes RDBs).

Similar to the results of design space exploration by varying the unit size of RDB, the number of RDBs also affects the memory system performance depending on the applications' memory access behaviors. Among all applications, *dedup, x264*, and *fluidanimate* are more sensitive than *swaptions, streamcluster*, and *raytrace* because the R/W ratios of them are lower than others. As described in Section 4.1, more write operations occupy more RDBs which mean the RDBs occupied from the read operations are evicted frequently. In our analysis, single write operation occupies up to 3 RDBs due to overlay window operations.

Overall, changing the number of RDBs shows relatively less effect on the memory access time compared to changing the unit size of RDB. We analyze that the cache captures

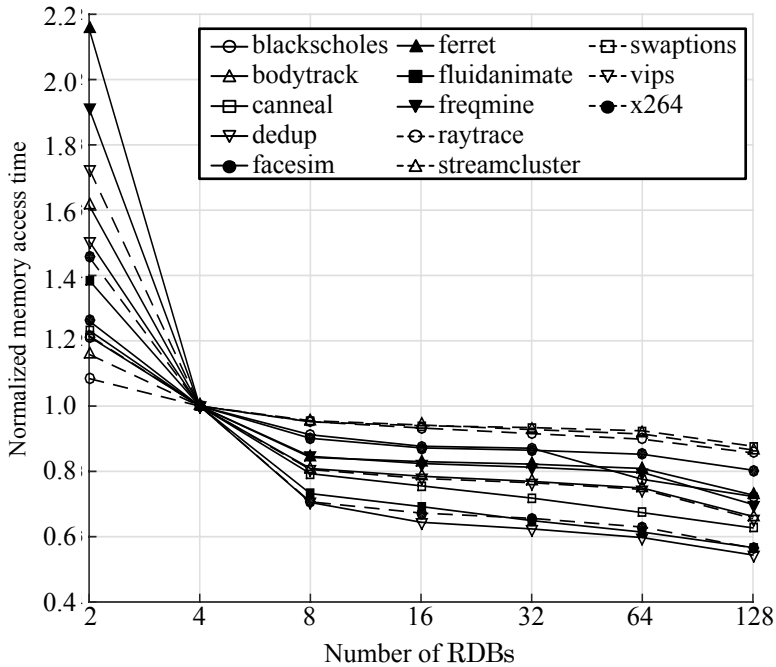


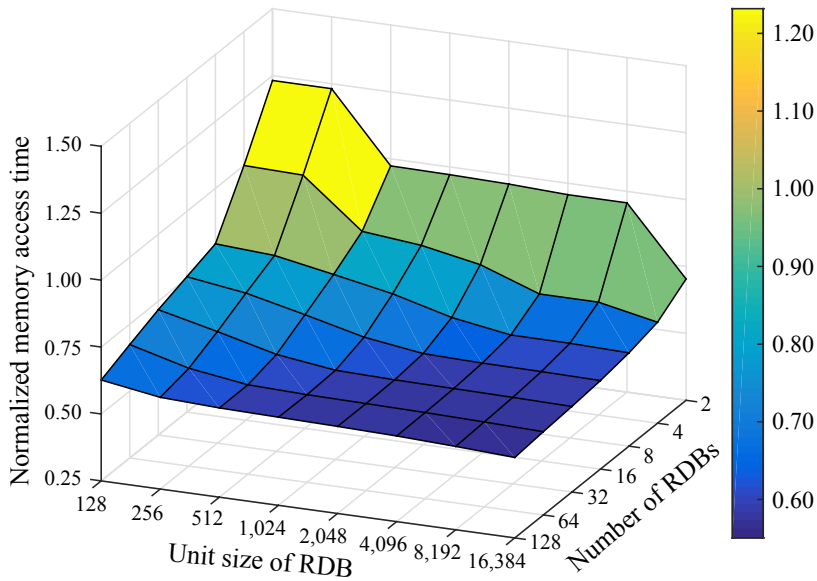
Figure 4.3: Memory access time varying on the number of RDBs. The unit size of RDB is fixed to 128 bytes for all configurations. The values of each memory access time are normalized to the memory access time of 4×128 bytes RDBs configuration.

the random memory accesses effectively.

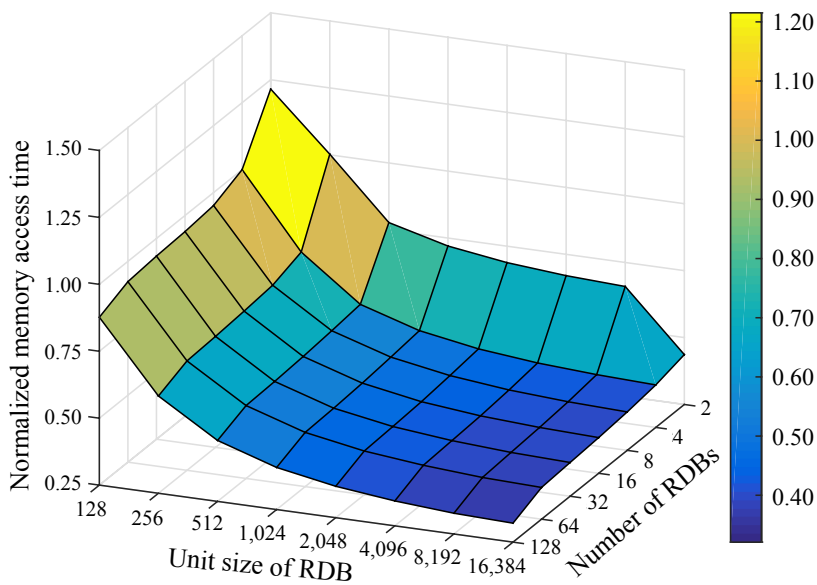
4.2.3 Unit size of row data buffer vs the number of row data buffers

We explore the design space of row buffer by increasing the unit size of RDB while the number of RDBs is fixed and changing the number of RDBs while the unit size of RDB is fixed in the previous subsections. We focus on the relation of memory access time and other design parameters such as RDB configuration, the number of cores, and the size of L2 cache in this section.

Figure 4.4 shows the memory access time varying on the RDB configurations. We select



(a) canneal



(b) swaptions

Figure 4.4: Normalized memory access time with different RDB configurations.

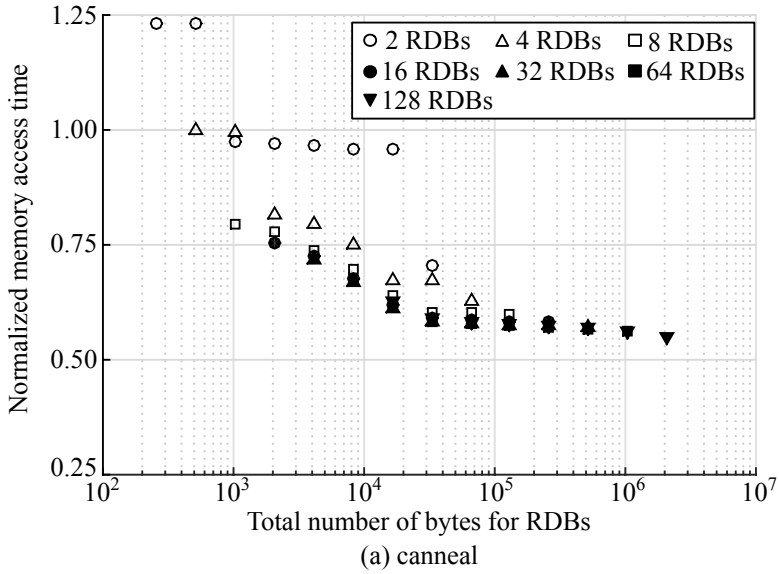
the two applications which are the representative application of random and sequential behaviors among the all other applications. We change the unit size of RDB from 128 to 16,384 bytes and the number of RDBs from 2 to 128. We normalize the memory access time of each configuration to the baseline configuration (4×128 bytes RDBs).

canneal has more random access pattern than sequential access pattern, so the RDB configuration that has higher number of RDBs shows less memory access time than the RDB configuration that has larger unit size of RDB if the total number of bytes for RDB is same, as shown in Figure 4.4(a). On the other hand, *swaptions* has more sequential access pattern than random access pattern. Figure 4.4(b) shows that increasing the unit size of RDB reduces more memory access time than increasing the number of RDBs for *swaptions*.

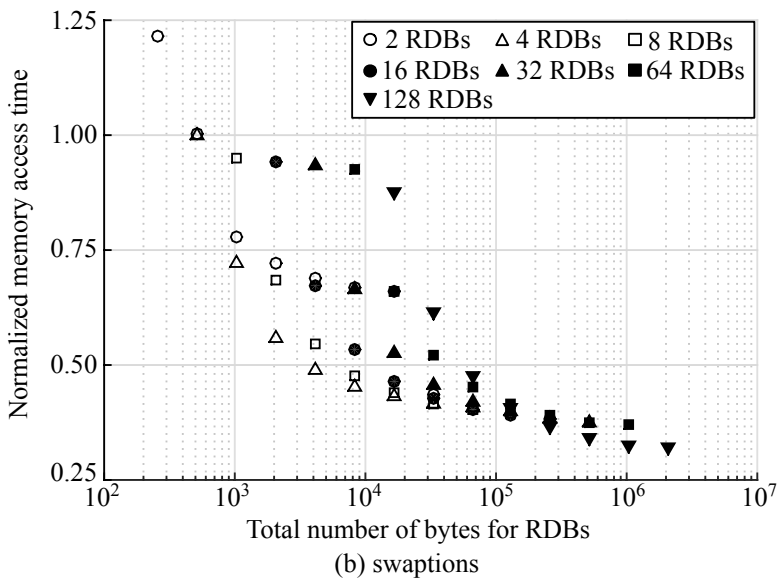
The memory access time decreases if we use more resources on row buffer, but it increases cost. Since the silicon area and the performance enhancement are not exchangeable, we derive a Pareto optimum under the same area constraint, which implies the maximum performance gain with a given constraint. We use the total number of bytes for RDBs to present the silicon area overhead where the complexity of the control logic is expected to be proportional to the unit size of RDB and the number of RDBs in general.

Figure 4.5 shows the variation of the memory access time with different RDB configurations under the same area constraint for two representative applications. We change the total number of bytes for RDBs from 256 bytes to 2,097,152 bytes. We normalize the memory access time of each benchmark with the different unit size and number of RDBs to the 4×128 bytes RDBs configuration.

The result presented in Figure 4.5 clearly shows that there are the Pareto optima of RDB configurations under the same area constraint. For instance, with 2,048 bytes of RDBs, 16×128 bytes configuration and 4×512 bytes configuration are Pareto optimum for *canneal* (Figure 4.5(a)) and *swaption* (Figure 4.5(b)), respectively. This analysis clearly shows

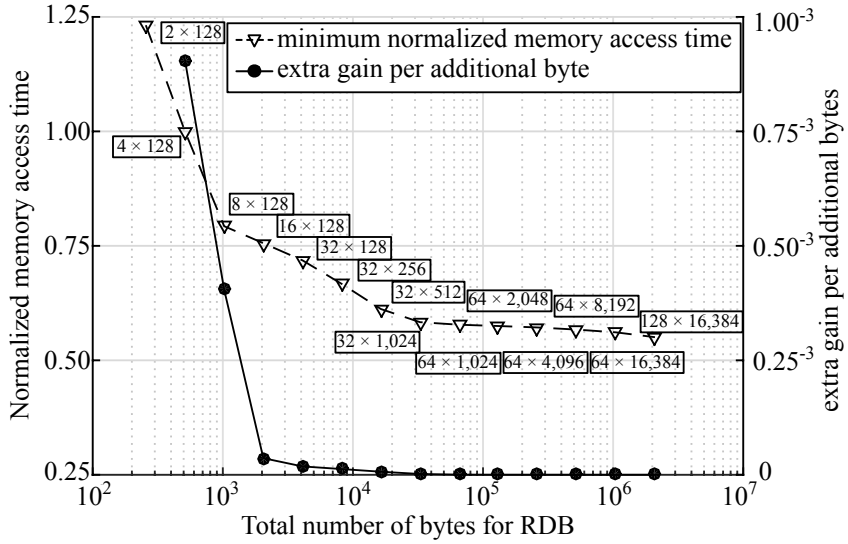


(a) canneal

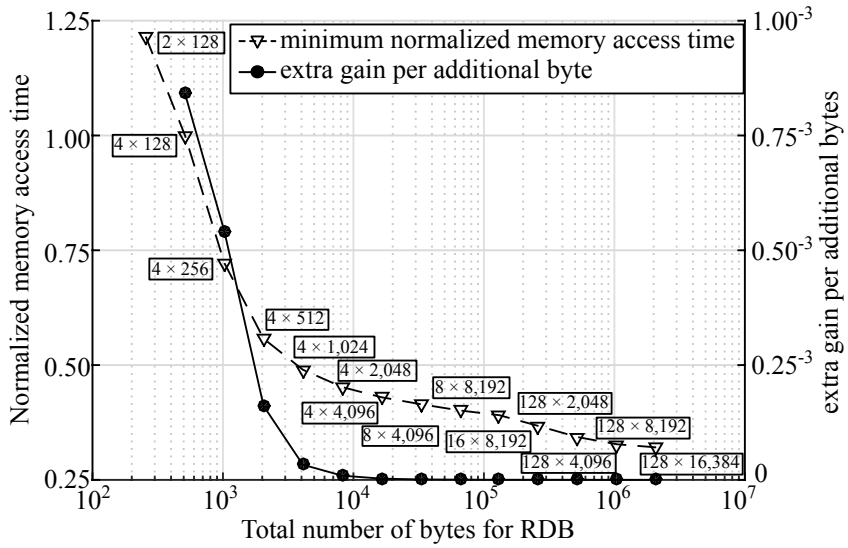


(b) swaptions

Figure 4.5: Memory access time comparison of different RDB configurations under the same area constraint.



(a) Random - Canneal



(b) Sequential - swaption

Figure 4.6: Optimal memory access time with Pareto optimum of RDB configurations under the same area constraint and extra gain per additional byte.

that the performance of the memory system is varying on the design of row buffer configuration even under the same area constraint. In our evaluation, we observed that the properly designed row buffer configuration enhances the memory performance up to 61.8% compare to the improperly designed row buffer configuration.

Figure 4.6 shows the extra performance gain per additional byte for two representative applications. The open inverted triangle with dash line in Figure 4.6 represents the Pareto optima row buffer configurations under the same area constraint. The closed circle with solid line represents the extra gain per additional byte if we choose the Pareto optima RDB configurations under the same area constraint. It shows the point that additional byte do not significantly affect the performance. Two applications have a different knee point because the memory access pattern is different.

We evaluate the memory access time variation according to the RDB configuration. The optimum RDB configuration varies on the memory access pattern of the applications. There are many design parameters that affect the memory access pattern of the applications but we focus on the number of cores and the size of shared L2 cache. We change the unit size of RDB from 128 to 16,384 bytes and the number of RDBs from 2 to 128. We normalize the memory access time of each configuration to the baseline configuration (4×128 bytes RDBs).

Figure 4.7 shows the normalized memory access time variation according to the number of cores. We select two representative applications, *anneal* and *swaptions*, and evaluate the memory access time with 4 cores, 8 cores, and 16 cores. We allocate a thread of application to each core. The number of memory access is proportional to the number of cores because the size of cache is fixed.

The memory access pattern of *canneal* with different number of cores changes significantly, as shown in Figure 4.7(a). The memory access time reduction with 4 cores does not

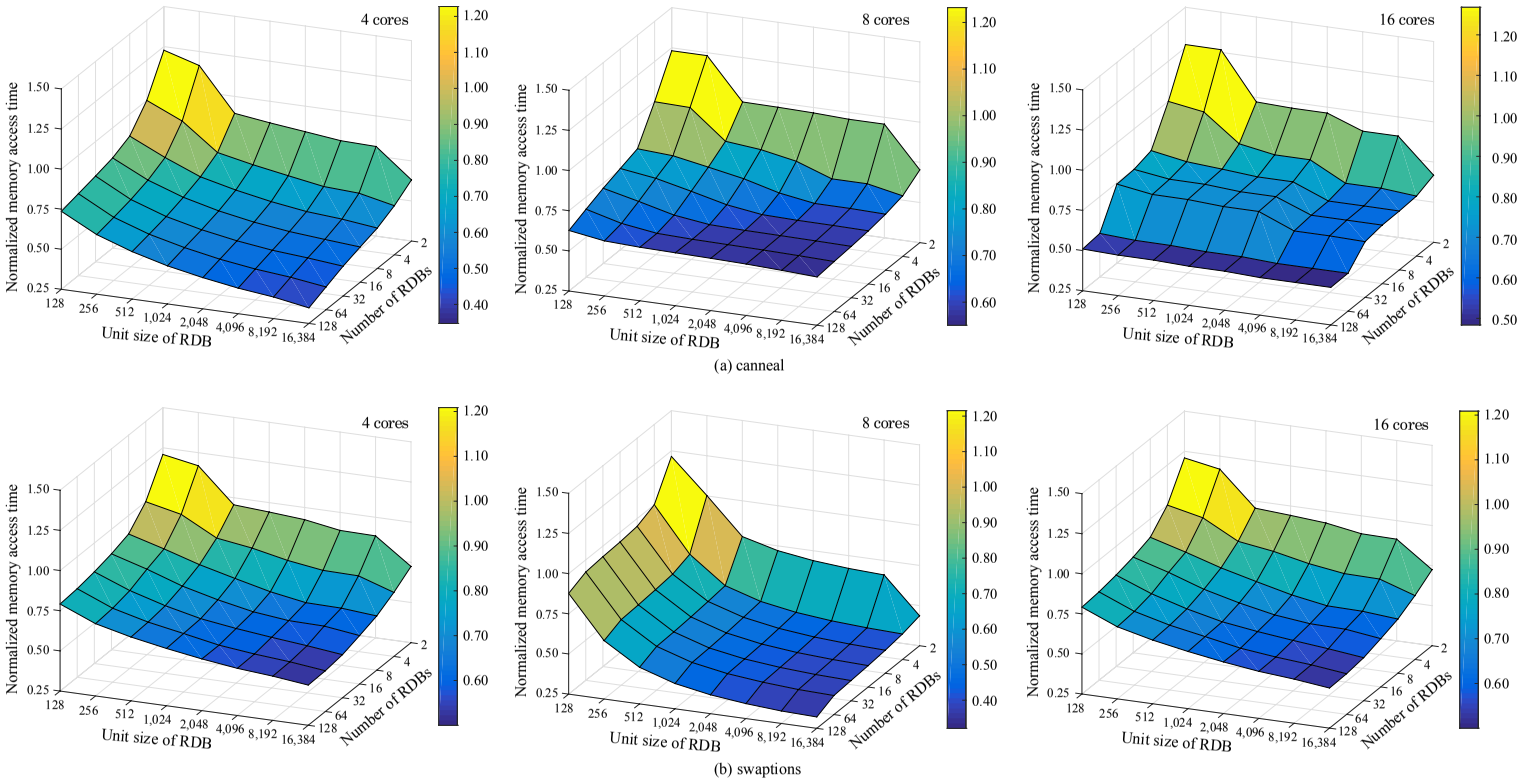


Figure 4.7: Normalized memory access time of *canneal* and *swaptions* with the number of core variation.

have knee point, and it is proportional to the unit size of RDB and the number of RDBs. The increment of total number of bytes for RDB does not reduce the memory access time with 8 cores significantly after the total number of bytes for RDBs exceeds 32,768 bytes. The memory access time with 16 cores reduces significantly when the unit size of RDB changes from 2,048 to 4,096 bytes and the number of RDBs changes from 32 to 64. The memory access time reduction of *swaptions* with 8 cores is not sensitive to the number of RDBs change, but that of *swaptions* with 4 cores and 16 cores are sensitive to the number of RDBs change, as shown in Figure 4.7(b). The number of cores variation changes the memory access pattern as similar as application changes.

The size of cache is another factor that can change the memory access pattern of the applications. We change the size of shared L2 cache from 1MB to 2MB and 4MB while keeping the number of way and the block size as same. The number of memory accesses decreases as the size of shared L2 cache increases. We analyze the variation of the memory access time reduction with two representative applications, *canneal* and *swaptions*.

The size of L2 cache does not change the memory access pattern of application significantly, as shown in Figure 4.8. The memory access time reduction due to increment of the number of RDBs affects more than that is caused by increasing the unit size of RDB as the size of shared L2 cache increases. As a result, the slope of surface is a little changed. Figure 4.8 also shows that the memory access time reduction decreases slightly as the size of shared L2 cache increases.

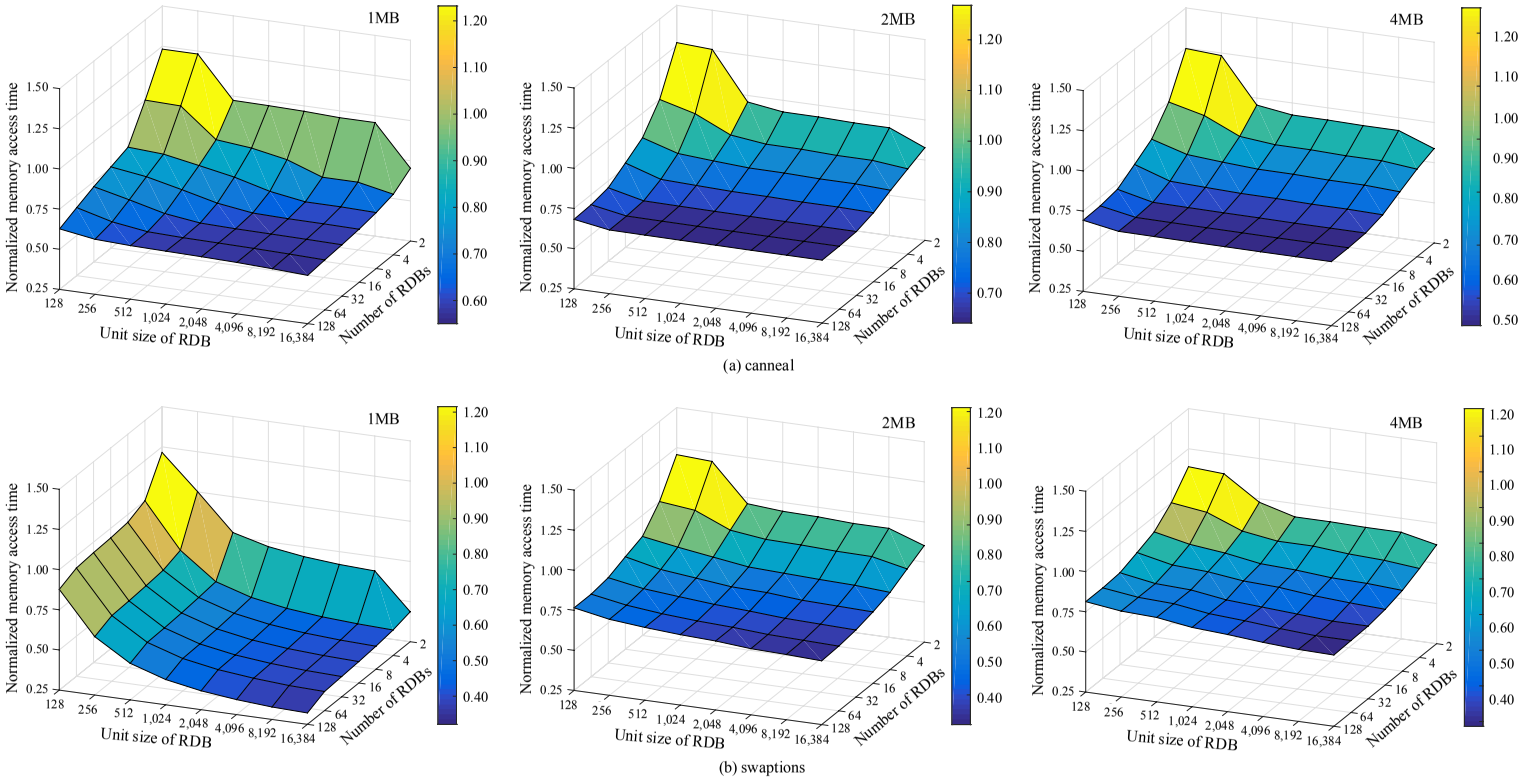


Figure 4.8: Normalized memory access time of *canneal* and *swaptions* with L2 cache size variation.

Chapter 5

System-Level Performance Optimization

This chapter proposes system-level performance optimization based on the clues from Chapter 4. First, we improves performance of memory system by reforming the way of interfacing LPDDR2-NVM. The memory controller omits the part of three-phase addressing if RAB or RDB has valid data. Second, we show the limitation of static row buffer architecture and propose a method that mimics reconfigurable row buffer architecture by managing row buffers proactively.

5.1 Address Phase Skipping

5.1.1 Motivation

In LPDDR2-NVM standard, the preactive phase or activate phase can be skipped if the RAB or RDB has valid data, as shown in Figure 5.1. Figure 5.1(a) shows a full addressing sequence. Without valid data in the RAB and RDB, the memory controller should perform

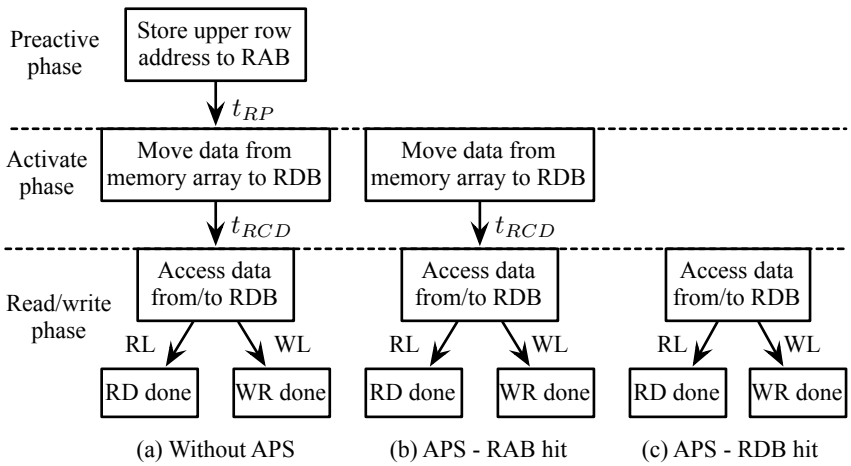


Figure 5.1: Operation of address phase skipping.

all three addressing phases. Figure 5.1(b) shows the case of a RAB hit. The addressing sequence can start from the activate phase, and we save the time from the preactive to the activate command, t_{RP}. This address phase skipping (APS) method has the largest time savings when the RDB already contains valid data, as shown in Figure 5.1(c). With valid data, time consuming activate-to-read/write command period, t_{RCD}, would be removed [74].

The incoming memory accesses due to cache misses in the processor makes a RDB hit because the unit size of RDB is bigger than the size of a single cacheline in the processor. The proposed method skips the part of the address phase when we observe RAB hits. This RAB hit is only applicable for the main memory system using LPDDR2-NVM. The proposed method does not require significant overhead because it utilizes the existing row buffers in LPDDR2-NVM.

5.1.2 Address phase and row buffer decision

We need an algorithm to manage row buffers to exploit the time advantage with APS because LPDDR2-NVM has different row buffer architecture compared to conventional

Algorithm 1: Address phase and address of row buffer decision algorithm

```
Data: access address

Result: address phase, address of row buffer

if Both RAB and RDB are valid, and lower row hit then
    Address phase ← READ/WRITE

    Address of row buffer ← hit RDB
else if RAB is valid, and upper row hit then
    Address phase ← ACTIVATE

    Address of row buffer ← LRU among hit RABs
else
    Address phase ← PREACTIVE

    if invalid RDB exist then
        Address of row buffer ← invalid RDB

    else
        Address of row buffer ← LRU among RDBs
```

DRAM, as described in Chapter 2. We propose a row buffer management method that consists of an address phase and row buffer decision algorithm plus a row buffer status management algorithm.

The memory controller should track the row buffer contents to decide that the part of addressing phase is skipped. The proposed method uses a content-addressable memory to store the address of the row buffer contents. When a new memory access comes, the memory controller determines the starting point of the addressing phase based on this information, as shown in Algorithm 1. We do not need the row buffer contents itself to determine the starting point of the addressing point, so the overhead of the APS is not significant. To utilizing the row buffer to efficiently skip the addressing phase, we need a row buffer replacement policy similar to the cache, where the number of the row buffer is limited to

Algorithm 2: Row buffer status management algorithm

Data: access type, access address, access data

Result: row address buffer, RABs status, RDBs status

if *access is accepted* **then**
    Row address buffer ← row address of access
    Status of selected RAB ← valid
    if *access is read in memory array* **then**
        Status of selected RDB ← valid
    else if *access is write in overlay window* **then**
        if *access starts cell programming, and the programming data is stored in the valid RDB* **then**
            Status of selected RDB ← invalid

4 or 8 in the LPDDR2-NVM standard. We select a victim based on the least recently used (LRU) algorithm, as described in Algorithm 1.

5.1.3 Row buffer status management

An activated row in conventional DRAM should be precharged before accessing a different row in the same bank, and thus the valid period of row buffer contents is determined from the state of DRAM. However, LPDDR2-NVM does not require a precharge operation to access a different row. The contents of RABs and RDBs are valid as long as power is supplied, so the validity of RABs RDBs should be tracked. The contents in the RDB are not valid any more when the write request updates the data in the memory array because the contents in the RDB are not updated automatically until the updated row is activated again. Algorithm 2 prevents this inconsistency problem by invalidating RDB while the part

Table 5.1: LPDDR2-NVM SODIMM parameters.

Capacity	512MB
Speed bin	LPDDR2-400
Preactive to Activate (t_{RP})	3 t_{CK}[1]
Activate to Read/Write (t_{RCD})	80ns
Read latency (RL)	3 t_{CK}[1]
Write latency (WL)	1 t_{CK}[1]
Burst length (BL)	8
Cell program time ($t_{program}$)	20,000ns
The number of RDBs	4
The unit size of RDB	128 bytes

[1] t_{CK} is a memory clock cycle (5ns at LPDDR2-400)

of the row address in RAB remains valid.

5.1.4 Experiments

We first verify the operation of the APS and evaluate memory access time reduction with LPDDR2-NVM SODIMM by using synthetic workloads, such as sequential read, sequential write, random read, and random write accesses. Each synthetic workload consists of 1,000,000 accesses of 32 bits read or write. We increase the address by 4 to generate the sequential workload. The XORShift32 is used to generate random workload. The synthetic workload operates at MicroBlaze without cache. The parameters of LPDDR2-NVM SODIMM are summarized in Table 5.1.

Table 5.2 shows the measured memory access time of the synthetic workloads. The

Table 5.2: Comparison of the measured memory access time.

Workload	without APS (ms)	with APS (ms)	Reduction (%)
Sequential read	189.99	82.81	56.41
Random read	190.00	189.38	0.33
Sequential write	25,237.58	25,198.58	0.15
Random write	26,163.62	26,103.17	0.23

APS does not show the significant memory access time reduction with sequential write and random write. The APS reduces memory access time by reducing the time spent on the interface. However, the cell program time of the PCM device used in LPDDR2-NVM SODIMM is about a hundred times longer than the interfacing time, in turn, the cell program time hides the effect of APS. The APS shows significant memory access time reduction with sequential read because 3.1% and 96.9% of sequential read occur RAB hit and RDB hit, respectively. The RAB hit and the RDB hit reduces about 3.3% and 56.7% of memory access time, respectively. The high RDB hit ratio in the sequential read leads to significant memory access time reduction. The memory access time reduction with random read is negligible because 3.1% of random read hits RAB, and the rest of memory accesses miss RABs and RDB.

The memory controller with APS has to keep the address of contents in RDBs to judge a RDB hit or miss. It also need to implement Algorithm 1 and 2. We implement those algorithms with combinational logic, so APS is implemented in the memory controller without additional time overhead. The memory controller uses a content-addressable memory to store the address of contents in RDBs, but the number of RDBs is only four and it requires row address only. Therefore, the APS is implemented with small area overhead. Table 5.3

Table 5.3: Comparison of memory controller area.

	without APS	with APS	Overhead (%)
Number of slice registers	11,647	11,850	1.74
Number of slice LUTs	10,787	11,165	3.50
Number of fully used LUT-FF pairs	4,792	4,966	3.63

shows the area comparison of memory controller without APS and APS. We use Kintex-7 XC7K325T-2FFG900.

We also evaluate memory access time with realistic trace using a system-level simulator described in Section 3.2. We use the trace of the PARSEC benchmark [73] from a Simics full-system simulator [72]. Table 5.4 shows the details of the simulated system. The parameter of LPDDR2-NVM SODIMM is used as the parameters of LPDDR2-NVM module except the cell program time. We use four types of cell program time: 20μs, 10μs, 1μs and 150ns. This assumption on the cell program time is not an optimistic because a research prototype from the industry takes 150ns to program a cell [15].

Figure 5.2 shows the memory access time reduction when the APS is applied. We assume that write access completes when the cell programming ends. We also assume that the NVM does not support a dual operation. All the values are normalized to the memory access time without the APS on each application.

The spatial locality and temporal locality of read access significantly affect the row buffer hit ratio, and this finally results in the memory access time reduction. The RAB and RDB hit ratio of read accesses, $r_{RD.RAB}$ and $r_{RD.RDB}$, varies from 9.3% to 69.2% and from 0.1% to 42.6%, respectively, as shown in Table 5.5. On the other hand, memory access of write access is not affected by the spatial locality of it because of the asymmetric charac-

Table 5.4: Configuration of the simulated system.

System	8 cores in-order processor
Processor	UltraSPARC-III+, 2GHz
L1 cache (Private)	I- and D-cache: 64KB, 4-way 64B block
L2 cache (Shared)	1MB, 4-way 64B block
LPDDR2-NVM main memory	1GB, LPDDR2-400, 64-bit wide
Preactive to Activate (t_{RP})	3 t_{CK}[1]
Activate to Read/Write (t_{RCD})	80ns
Read latency (RL)	3 t_{CK}[1]
Write latency (WL)	1 t_{CK}[1]
Burst length (BL)	8
Cell program time ($t_{program}$)	150 $\sim$ 20,000ns
The number of RDBs	4
The unit size of RDB	128 bytes

[1] t_{CK} is a memory clock cycle (5ns at LPDDR2-400)

teristic of accessing flow. The row buffer hit ratio during the overlay window access for a write operation is kept almost high regardless of the locality. The reason is that write access of NVM is translated into the several consecutive overlay window accesses under the LPDDR2-NVM standard, and the address of overlay window does not change. The RAB and RDB hit ratio of overlay window accesses, $r_{OW.RAB}$ and $r_{OW.RDB}$, varies from 17.9% to 33.9% and from 36.2% to 78.4%, respectively, as shown in Table 5.5. This implies that the proposed APS method works more efficiently for write accesses regardless of the application. The simulation results also show that the effect of cell programming time variation on

Table 5.5: The RAB and RDB hit ratio of read and write accesses with APS.

Application	$r_{RD.RAB}$	$r_{RD.RDB}$	$r_{OW.RAB}$	$r_{OW.RDB}$
blackscholes	36.8	13.2	20.7	54.2
bodytrack	39.6	27.2	23.4	68.3
canneal	9.3	0.1	33.9	36.2
dedup	28.5	13.0	27.6	63.3
facesim	58.0	34.5	23.3	68.1
ferret	43.9	37.6	17.9	78.4
fluidanimate	29.1	4.8	31.4	58.8
freqmine	39.6	34.0	19.7	75.1
raytrace	69.2	20.6	26.2	38.6
streamcluster	51.7	33.5	24.3	58.5
swaptions	54.2	42.6	20.8	75.9
vips	44.5	25.0	21.2	75.5
x264	27.8	17.2	27.6	57.3

the RAB and RDB hit ratio is negligible.

The enhancement of APS is not significant in Figure 5.2(a)–20μs programming time– because the relatively long cell program time hides the effect of the proposed method, although we observe an average 65.8% and 91.0% row buffer hit ratio on read accesses and overlay window accesses, respectively. The effect of the proposed method increases up to 41.8% where the cell program time reduces to 150ns, as shown in Figure 5.2(b), 5.2(c), and 5.2(d). The applications with high R/W ratio such as *facesim, raytrace, streamcluster, and swaptions* show significant memory access time reduction when the cell programming

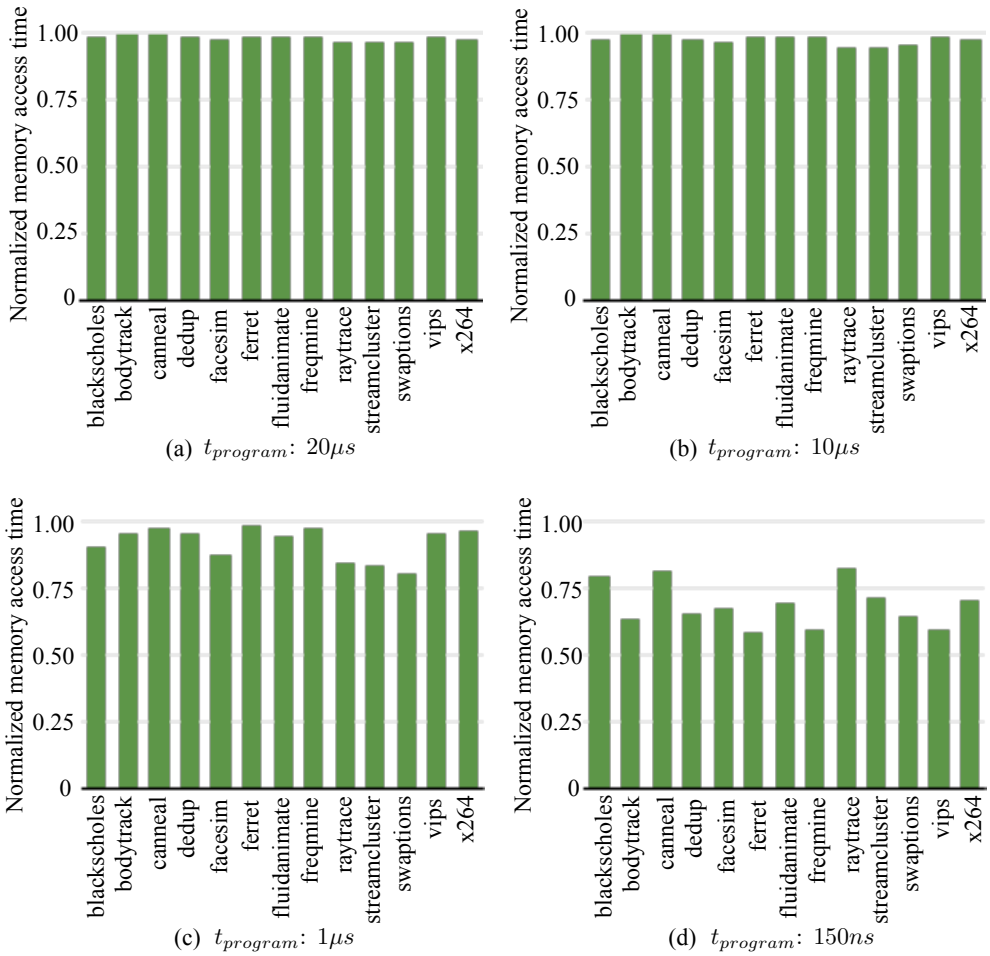


Figure 5.2: Memory access time of the APS normalized to memory access time without APS on each application.

time is 1μs, as shown in Figure 5.2(c). The cell program time in these applications does not dominate memory access time because high R/W ratio means that these applications have much read accesses than write accesses.

The proposed APS method reduces about 70% of single memory access time when the RDB hit occurs during read and overlay window access. The overlay window access time becomes similar to the cell program time if the cell program time becomes 150 ns. In such a case, a single write access takes about 7 times longer than a read access without the proposed APS method, whereas write access time with the APS method reduces to 5 times the read access time. This contributes to reduce the memory access time significantly. Figure 5.2(d) shows that the APS reduces the memory access time of the applications with low R/W ratio such as *bodytrack, dedup, ferret, fluid animate, freemen, vips, and x264* significantly.

Figure 5.3 shows the effect of APS on the total execution time. The blue part of the bar is the CPU execution time and the green part of the bar is the memory access time. The proposed APS does not reduce the CPU execution time. The total execution time is reduced up to 2.1% if the cell program time is 20μs, as shown in Figure 5.3(a). The effect of APS increases up to 25.5% as the cell program time is reduced, as shown in Figure 5.3(b), 5.3(c), and 5.3(d).

We analyze the potential of the proposed methodology with respect to the variation of PCM cell program time, as shown in Figure 5.4. We use two applications (*raytrace* and *vips*) that are read dominant and write dominant among all the applications. In both applications, our APS method dramatically reduces the memory access time if the PCM cell program time is reduced. The reduced memory access times of *raytrace* and *vips* with the APS are, respectively, 18.4% and 44.9% of the memory access time without the APS at 150ns cell program time. In addition, the APS shows more effect on write access than on

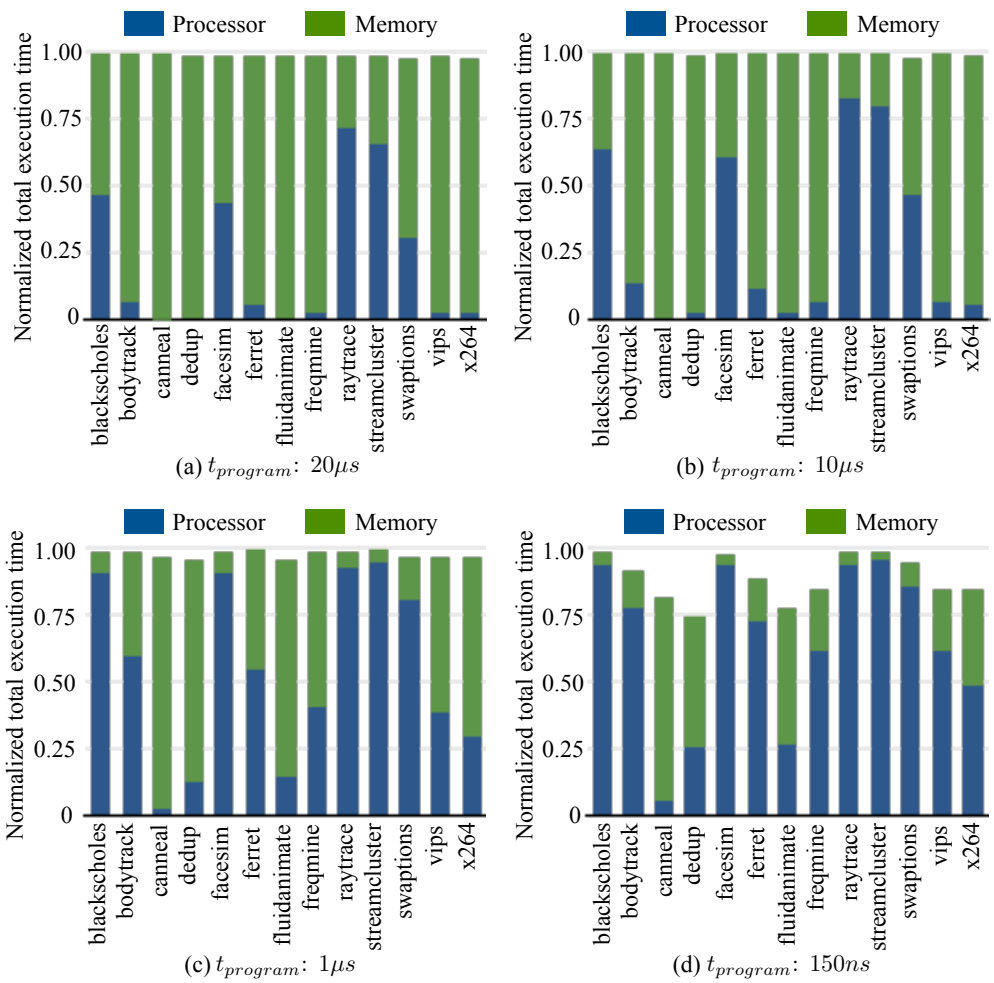


Figure 5.3: Total execution time of the APS normalized to total execution time without APS on each application.

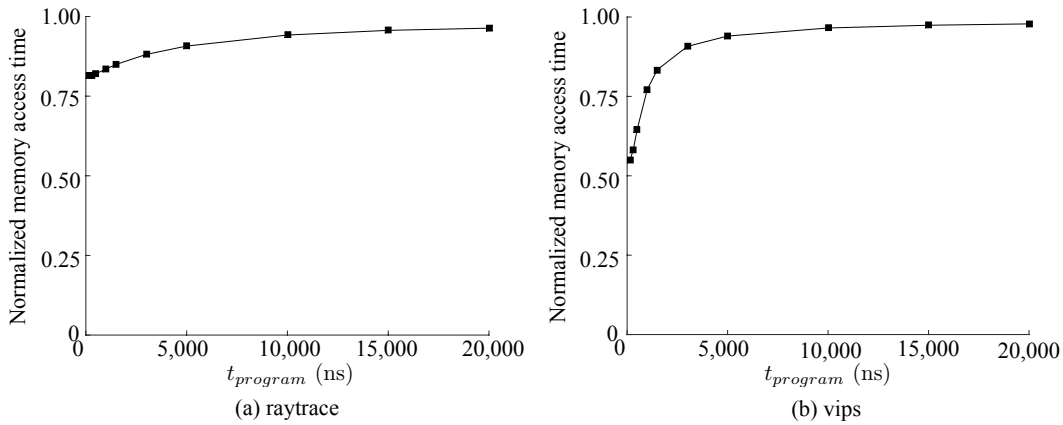


Figure 5.4: Sensitivity analysis of memory access time varying on PCM cell program time. The values of each time parameter are normalized to the memory access time without APS.

read access if the PCM cell program time is reduced.

5.2 Proactive Row Buffer Management

5.2.1 Motivation

As described in Chapter 2, the row buffer architecture of LPDDR2-NVM is different from that in conventional DRAMs where the row buffer architecture is fixed at the design time. On the other hand, the LPDDR2-NVM standard specifies more flexibility in designing and managing the row buffer architecture. For example, LPDDR2-NVM compatible device can have several row buffers which are not tightly coupled with the address. In addition, the memory controller may use any row buffer among several row buffers for any array location. This flexible architecture enables us to design various row buffer management policies considering the access patterns of the applications.

In designing row buffer architecture, determining the unit size of RDB and the number of RDBs are as important as determining the total number of bytes for RDB. Figure 5.5

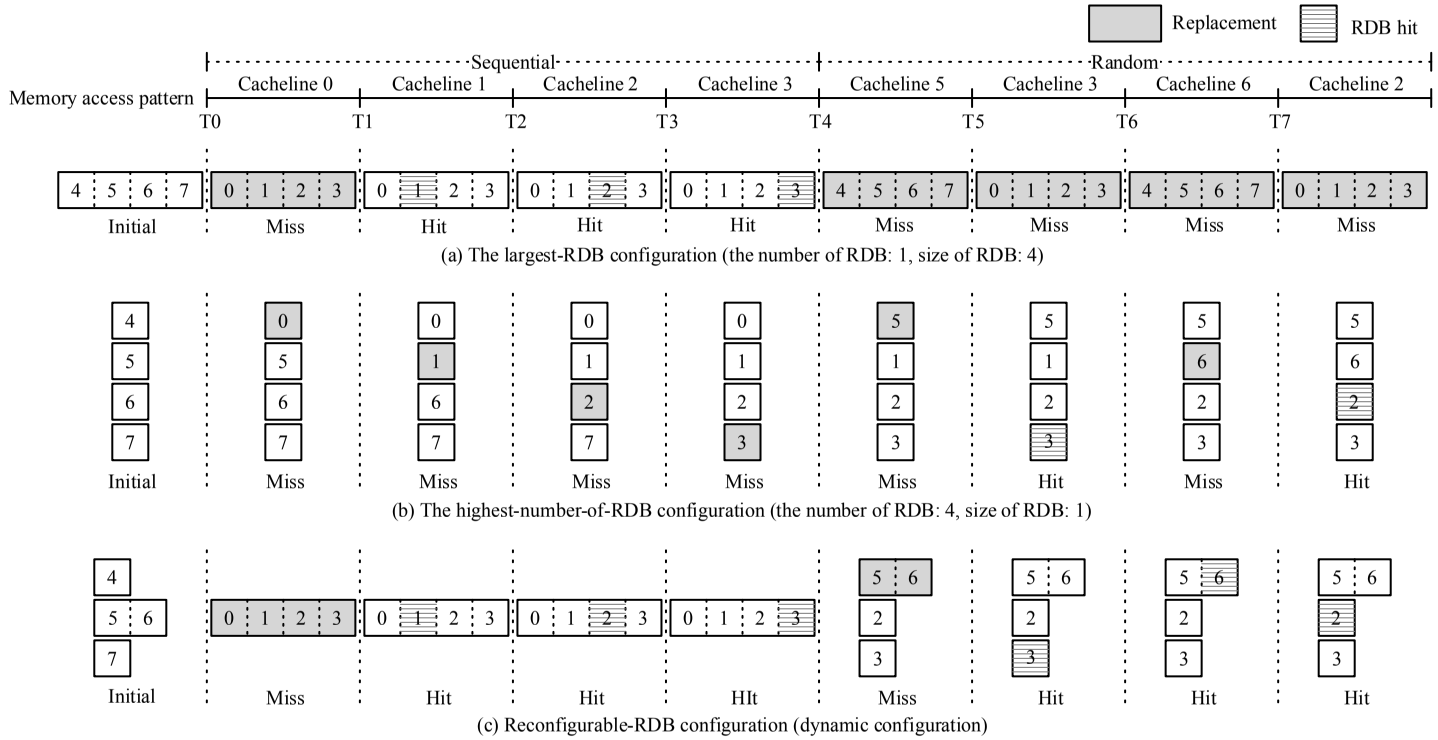


Figure 5.5: RDB hit ratio varying on row buffer architecture and memory access patterns.

shows a motivational example of this work. We simply compare the number of RDB hits on the three different configurations; (a) The largest-RDB configuration, (b) The highest-number-of-RDB configuration, and (c) reconfigurable-RDB configuration. In the figure, the box with thick solid line means one physical RDB and one physical RDB consists of one(or more) basic units – the box with dotted line. The size of one basic unit is equal to the size of one cacheline. The largest-RDB configuration has only one RDB which consists of 4 basic units while the highest-number-of-RDB configuration has four RDBs where each RDB size is equal to one basic unit. The example memory access patterns are showed on the top of the figure. The first half of memory access pattern is sequential while the second half of pattern is random. A grayed-box and horizontally-lined-box represents a RDB hit and a RDB miss, respectively. For fair comparison, all three configurations start with same initial state – Cachelines 4, 5, 6, and 7 are stored in the RDBs.

In the largest-RDB configuration, the request of Cacheline 0 incurs a RDB miss at time T0. This RDB miss evicts all cachelines in the RDB, and then Cachelines 0 to 3 are fetched from the NVM array, as shown in Figure 5.5(a). Since the next three memory accesses are sequential, all three requests incur RDB hits. However, remaining memory access pattern at time T4 to T7 incurs RDB misses again because of only one RDB. In total, 5 RDB misses and 3 RDB hits are occurred during the memory accesses.

In contrast, the highest-number-of-RDB configuration handles a random memory access pattern efficiently because one RDB has only one cacheline. However, it is very weak to sequential memory access pattern. The requests of Cachelines 0 to 3 continuously incur RDB misses at time T0 to T3, as shown in Figure 5.5(b). In total, we observe 6 RDB misses and 2 RDB hits.

Both the largest-RDB configuration and the highest-number-of-RDB configuration provide limited capability to the given memory access pattern in the motivational example.

Each configuration has advantages and disadvantages according to the characteristics of memory access pattern. The characteristics of memory access pattern changes if operating application changes and/or time goes. Changing the RDB configuration dynamically increases RDB hit with any kind of memory access pattern, as shown in Figure 5.5(c). It reduces RDB misses by reconfiguring the row buffer as one RDB with four cacheline size and replacing it with four consecutive cachelines at T0. It makes the next three memory accesses as RDB hits. The reconfigurable-RDB configuration modifies its RDB configuration as one RDB with size of two cacheline and two RDB with size of one cacheline at T4. This reconfiguration turns the remains of random memory accesses from T5 to T7 as RDB hit, as shown in Figure 5.5(c). In total, 2 RDB misses and 6 RDB hits are occurred.

This dynamic reconfiguration shows the best RDB hit ratio with information of incoming memory access pattern. It is important to predict the characteristics of memory access accurately because reconfiguring RDB architecture with wrong information is possible to incur more RDB misses.

5.2.2 Proactive row buffer control policy

The dynamic reconfiguration of row buffer enable us to increase RDB hit ratio by changing its RDB configuration based on the information of incoming memory access pattern. However, the RDB configuration is fixed at the design time, and it does not support reconfiguration usually. We propose a method that works in the memory controller to mimic a reconfigurable RDB architecture. It predicts the characteristics of incoming memory access pattern to increase RDB hit ratio.

In addition to that, write operation in LPDDR2-NVM incurs several overlay window accesses, as described in Section 2.2. The memory access time reduces significantly if RDB hit ratio of the overlay window accesses increases. We propose a simple method that

increase RDB hit ratio with negligible overhead in the memory controller.

Prefetch technique in cache memories proactively moves a cacheline before it is requested during the time interval of memory accesses. For the processor, it seems that the size of one cacheline increases if the prefetch technique moves the successive cacheline to the cache before it is requested. It means that the configuration of cache changes at runtime without hardware modification. Similarly, we propose a row buffer prefetch technique that moves data to RDB before it is requested during the idle time between memory accesses. It works as increasing the unit size of RDB by reconfiguring the logical configuration of RDB without physical modification.

Row buffer prefetch moves the data based on the prediction. It is implemented in the memory controller. The memory controller predicts the characteristics of incoming memory access pattern based on the history. If the memory controller predict that the characteristics of incoming memory access pattern is a sequential, it inserts a row buffer prefetch command to a command queue.

It is important to predict the characteristics of incoming memory access pattern correctly. Row buffer prefetch replaces data of one RDB due to limited number of RDB. It may happen that the processor requests the abandoned data instead of the prefetched data if the prediction fails. It turns a RDB hit to a RDB miss, thus increases memory access time. The latency of memory access also increases if the row buffer prefetch does not finish until the arrival of the memory access from the processor.

We do not consider priority of memory access from the processor, and give a high priority to row buffer prefetch request than normal memory access. Row buffer prefetch requires less time than normal memory access because it does not execute read/write phase of three-phase addressing. It also have a chance to reduce the time consumption of incoming memory access if the prediction is accurate.

Some commercial DRAM controllers offer memory access reordering feature to increase row buffer hit ratio. It changes the order of memory access in the memory controller using reordering buffer, and returns the results to the processor as in-order. Reordering is able to increase RDB hit ratio of LPDDR2-NVM. Reordering is effective when the command queue has a number of memory accesses while row buffer prefetch uses idle time of the memory controller. Row buffer prefetch is orthogonal to reordering. However, we do not consider reordering because we focuses on the relation between the characteristics of memory access pattern and the RDB configuration.

We propose a system-level row buffer control policy that improve the memory system with LPDDR2-NVM using row buffer prefetch through the rest of this section.

5.2.2.1 Tagged row buffer prefetch

The tagged row buffer prefetch, T_{pre}, is similar to a tagged prefetch in a cache. It sets a bit on the RDB tracking table when a RDB is activated, as shown in Figure 5.6(a). This bit is cleared when the RDB is first accessed, and then the row buffer prefetch is requested to move the successive data to RDB, as shown in Figure 5.6(b). T_{pre} should evict one RDB to prefetch data, and it selects a victim based on the same replacement policy that is used when a RDB miss occurs.

T_{pre} moves data to the RDB aggressively. It is based on the assumption that memory access pattern will be a sequential if RDB is accessed more than twice. This assumption is justified by that the unit size of RDB is larger than the size of cacheline.

The row buffer prefetch is requested for read accesses only because a write memory access is translated into several overlay window access in LPDDR2-NVM. The address of overlay window does not change according to the address of the write memory access. This address translation makes the write memory access as random memory access even though

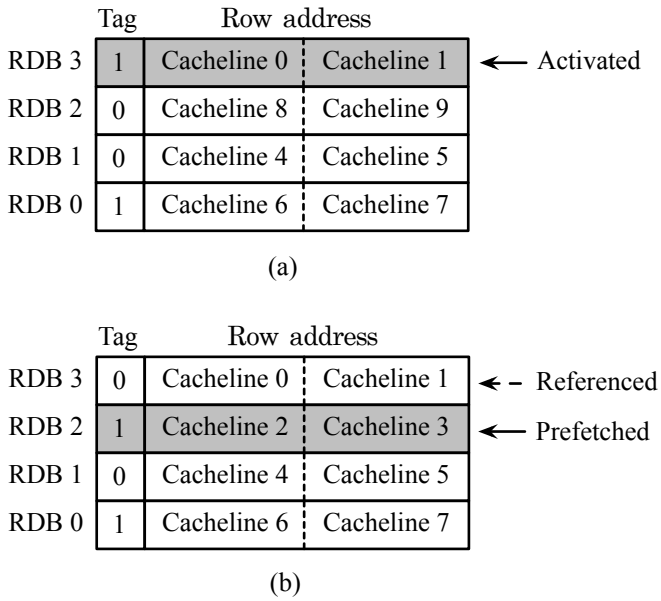


Figure 5.6: Operation of the tagged row buffer prefetch.

it is a part of sequential memory access pattern. Moreover, the overlay window accesses are executed seamlessly without idle time, so the row buffer prefetch does not reduce memory access time.

5.2.2.2 Multiple row buffer prefetch

Row buffer prefetch is initiated by a prediction, and accuracy of the prediction determines the reduction of memory access time. Unnecessary row buffer prefetch increases memory access time because it is possible to turn a RDB hit to a RDB miss and it blocks the memory access from the processor until the row buffer prefetch is done. We propose a technique to prevent unnecessary row buffer prefetches using two-bits saturating counter.

We use a two-bits saturating counter to predict the characteristics of incoming memory access pattern. This saturating counter changes a mode according to the recent activity of RDB, as shown in Figure 5.7. When a RDB hit occur, it decides that the incoming memory

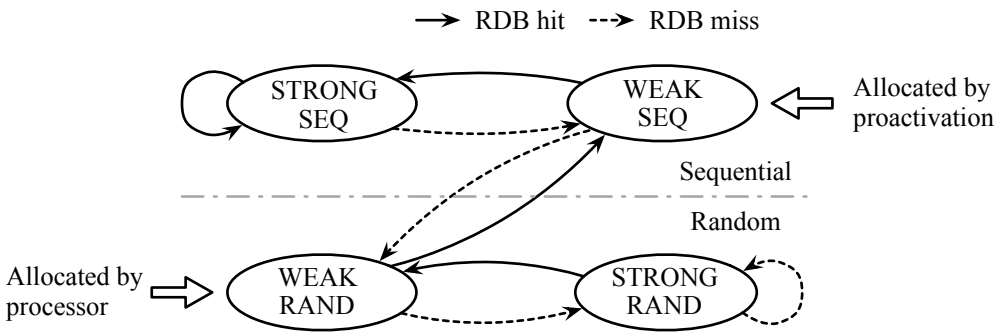


Figure 5.7: Mode transition and allocation of two-bit saturating counter for prediction of incoming memory accesses.

access will be sequential memory access because the size of RDB is bigger than the size of a single cacheline in the processor. It promotes the mode of saturating counter if it is not saturated. On the other hand, it considers that the incoming memory access will be random memory access if a RDB miss occurs. It demotes the mode of saturating counter. It requests a row buffer prefetch when a RDB hit occurs and the incoming memory access is sequential.

It is possible to use a global saturating counter for prediction, and it prevents unnecessary row buffer prefetches. However, a prediction of the global saturating counter is able to fail because the global saturating counter changes its mode quickly. When the memory access pattern is made up of sequential memory access patterns and random access patterns, a RDB miss due to random access demotes the mode of the global saturating counter, in turn, it predicts a sequential memory access as a random memory access. The global saturating counter tracks only one sequential memory access pattern, so it does not works well if several sequential memory access patterns are mixed.

Multiple row buffer prefetch, M_{pre}, uses multiple saturating counters as same as the number of RDBs to avoid this miss prediction. Multiple saturating counters are able to

track sequential memory access patterns up to the number of RDBs. When a RDB hit occurs, M_{pre} promotes mode of saturating counter that tracks the hit RDB only. It does not demote the mode of saturating counters that track other RDBs because it makes that a saturating counter predicts a sequential memory access pattern as a random. A RDB miss implies that it is not a part of sequential access patterns those are tracked by the saturating counters. M_{pre} demotes mode of all saturating counters at once.

We have to decide an initial mode of the saturating counter when new data is allocated to the RDB. RDB allocation is caused by the processor or a row buffer prefetch. The incoming memory access will be sequential if a row buffer prefetch allocates RDB. M_{pre} assigns WEAK SEQ mode to the tuple that tracks RDB allocated by the row buffer prefetch, as shown in Figure 5.7. The other case, it assigns WEAK RAND mode to the tuple, as shown in Figure 5.7. The memory access allocated by the processor misses all RDBs, so the incoming request will be a random memory access.

To skip the addressing phase, the memory controller should keep the address of data in RDBs and validity of RDBs. The overhead of tracking the data in RDBs is not significant because it is done by storing only row addresses. In addition to that, M_{pre} adds two-bits saturating counters to the tracking table. The saturating counters predict the spatial correlation between the stored data in each RDB and incoming memory access. This tracking table also enables us to track the temporal correlation between the stored data in RDB and incoming memory access. The row address of data in the RDB is evicted when the data in each RDB is replaced by new memory access or row buffer prefetch. It means that the evicted data does not have enough temporal correlation with incoming memory access.

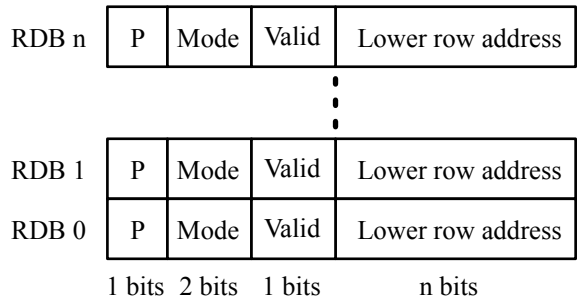


Figure 5.8: Row data buffer tracking table.

5.2.2.3 Multiple row buffer prefetch with overlay window pinning

As described in Chapter 2, the write memory access is translated into several overlay window accesses, and the address of overlay window is not altered according to the target address of write access. Therefore, write memory access causes dense overlay window accesses which access specific address. If a memory access pattern consists of read access and write access within a short period, RDBs contain overlay window have a high probability of RDB hit. Thus, RDBs contain overlay window have less probability that to be selected as victim by replacement policy, LRU, than RDBs contain memory array. However, it is possible to happen that RDBs contain overlay window are selected as victim even though incoming memory access pattern contains write access due to the limited number of RDBs.

To avoid this situation, we propose a simple method, overlay window pinning, that proactively manages RDBs contains overlay window efficiently based on the prediction. It dedicates RDBs for overlay window access if incoming memory access pattern has write accesses. It uses a weighted moving average, $W_{mv.avg}$, for prediction. It has a negligible area overhead because it needs a register with simple combinational logic and one bit for each RDB in row data buffer tracking table to indicate that RDB is pinned, i.e., RDB is not selected as victim, as shown in Figure 5.8.

Algorithm 3: Proactive RDB control policy of $M_{pre} + OW$

Data: information of access

Result: address phase, mode, row buffer prefetch, P bit

if *access is WRITE* **then**
  shift and evaluate $W_{mv.avg}$

if *access is HIT on* RDB_i **then**
  if *access is READ* **then**
    promote MODE of RDB_i **if** *MODE of* $RDB_i >=$ *WEAK SEQ and next data is not exist in RDB* **then**
      request row buffer prefetch
  else
    if $W_{mv.avg} >$ *WRITE THRESHOLD and access is not PROGRAM BUFFER* **then**
      P bit of $RDB_i \leftarrow$ SET
    else
      P bit of $RDB_i \leftarrow$ CLEAR
  ADDRESS PHASE = READ\WRITE
else
  for *All* RDB_i **do**
    demote MODE of RDB_i
  find RDB_{victim} that is not pinned based on LRU
  allocate access to RDB_{victim}
  ADDRESS PHASE $\leftarrow$ PREACTIVE
  MODE of $RDB_{victim} \leftarrow$ WEAK RAND
  if $W_{mv.avg} >$ *WRITE THRESHOLD and access is not PROGRAM BUFFER* **then**
    P bit of $RDB_{victim} \leftarrow$ SET
  else
    P bit of $RDB_{victim} \leftarrow$ CLEAR

if *row buffer prefetch is reqeusted* **then**
  find RDB_{victim} that is not pinned based on LRU
  allocate access to RDB_{victim}
  ADDRESS PHASE $\leftarrow$ PREACTIVE
  MODE of $RDB_{victim} \leftarrow$ WEAK SEQ

The proposed method evaluates $W_{mv.avg}$ with every memory access and pins a RDB for overlay window when the value of $W_{mv.avg}$ exceeds a predefined threshold, but it does not pin all RDBs that contain overlay window. The evaluated system uses a buffered overwrite command to program cells.Within the overlay window accesses, program buffer accesses show a low RDB hit ratio because the address of program buffer access changes according to the address of write access. Therefore, the proposed method does not pin the RDB contains program buffer.

We study several proactive row buffer control policies. T_{pre} increases RDB hit ratio of read access by making a larger effective unit size of RDB through row buffer prefetches. M_{pre} reduces memory access time by preventing unnecessary row buffer prefetches. Overlay window pinning increases RDB hit ratio of overlay window access while it sacrifices little RDB hit ratio of read access. These methods manage row buffer proactively based on the prediction. We propose a heuristic that controls the row data buffers proactively with combination of above methods, as shown in Algorithm 3. Multiple row buffer prefetch with overlay window pinning, $M_{pre} + OW$, is implemented in the memory controller without modification of the memory device.

5.2.3 Experiments

We evaluate the memory access time reduction of the proposed heuristic using a cycle-accurate trace-driven SystemC simulator described in Chapter 3. We use the trace of the PARSEC benchmark [73] from a Simics full-system simulator [72]. The details of the simulation setup are summarized in Table 5.6.

We compare the memory access time of static optimum RDB configuration, APS, T_{pre}, M_{pre}, and $M_{pre} + OW$. We measure the memory access time using latency at the processor. The static optimum RDB configuration is that the RDB configuration has a minimum

Table 5.6: Configuration of the simulated system.

System	8 cores in-order processor
Processor	UltraSPARC-III+, 2GHz
L1 cache (Private)	I- and D-cache: 64KB, 4-way 64B block
L2 cache (Shared)	1MB, 4-way 64B block
LPDDR2-NVM main memory	1GB, LPDDR2-800, 64-bit wide
Preactive to Activate (t_{RP})	3 t_{CK}[1]
Activate to Read/Write (t_{RCD})	120ns
Read latency (RL)	6 t_{CK}[1]
Write latency (WL)	3 t_{CK}[1]
Burst length (BL)	8
Cell program time ($t_{program}$)	150ns
The number of partitions	16

[1] t_{CK} is a memory clock cycle (2.5ns at LPDDR2-800)

memory access time among the possible RDB configurations. It is determined for each application through the design space exploration. The proposed heuristics work as increasing the effective unit size of RDB dynamically but not increasing the number of RDBs. Therefore, the heuristics work with the-highest-number-of-RDB configuration in each total number of bytes for RDBs. Table 5.7 shows the physical RDB configuration of each heuristics.

The memory controller has a write data queue so the processor ends write operation when the data reaches the write data queue. We assume that the memory device supports dual operation of LPDDR2-NVM. It enables read access in other partitions during cell pro-

Table 5.7: Physical RDB configuration of each heuristic.

Static optimum RDB configuration	2×512, 4×256, or 8×128 bytes
APS	8×128 bytes
T_{pre}	8×128 bytes
M_{pre}	8×128 bytes
$M_{pre} + OW$	8×128 bytes

gramming in a partition. It avoids the increase of read latency due to the long cell program time. However, the read latency increases when the processor try to read a partition that is cell programming because the read access has to wait until the cell programming is done.

Figure 5.9 shows memory access time of APS, T_{pre}, M_{pre} and $M_{pre} + OW$. The memory access time of each configuration is normalized to the static RDB configuration. The APS takes on average 9.2% more time than the static optimum RDB configuration. APS reduces memory access time when a RAB or a RDB hit occurs. However, static RDB configuration shows more RDB hit ratio than APS in the most of application because the RDB configuration is optimized for RDB hit. T_{pre} usually takes more memory access time than the static optimum configuration in the applications except *bodytrack, facesim, swaptions,* and *vips*. The row buffer prefetch makes the effective unit size of RDB larger, in turn, it reduces memory access time with sequential memory access pattern. However, the heuristic of T_{pre} prefetches a row buffer aggressively. It causes unnecessary row buffer prefetches, in thus, increases memory access time on average 9.0%. M_{pre} prevents unnecessary row buffer prefetches efficiently, so it reduces memory access time in the most applications except *fluidanimate* and *raytrace*. The average memory time reduction of M_{pre} is 11.3%. Figure 5.9 shows that $M_{pre} + OW$ predicts the characteristics of incoming memory access

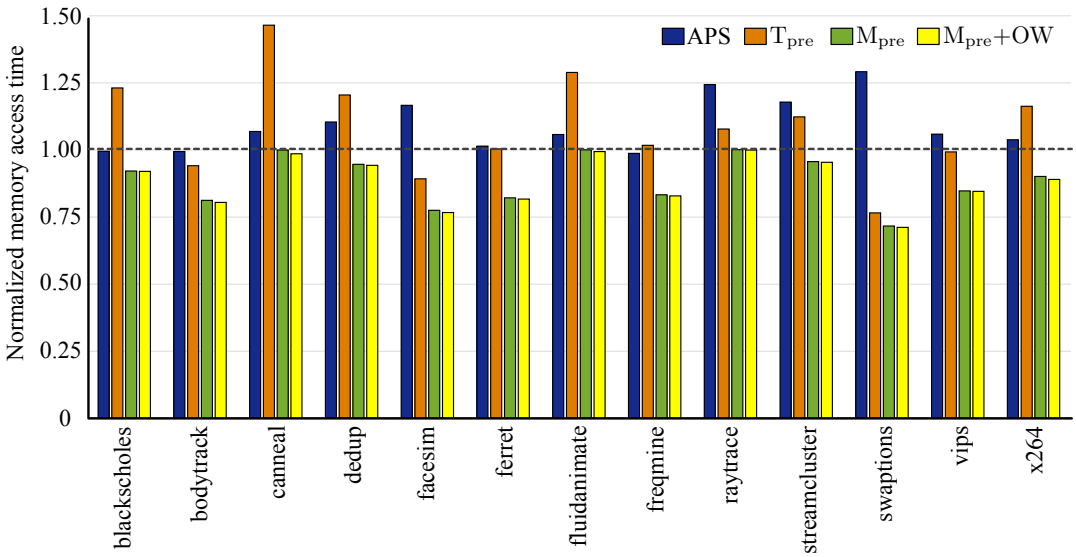


Figure 5.9: Comparison of memory access time normalized to the static optimum RDB configuration.

pattern more accurately. $M_{pre} + OW$ shows on average 11.8% and up to 28.8% memory access time reduction.

Table 5.8 shows the hit ratio comparison of the static optimum RDB configuration, APS, T_{pre}, M_{pre}, and $M_{pre} + OW$. We distinguish the RDB hit ratio of read access, r_{RD}, and overlay window access, r_{OW}. APS has not only a RDB hit, but also a RAB hit. We uses $r_{RD.RAB}$, $r_{RD.RDB}$, $r_{OW.RAB}$, and $r_{OW.RDB}$ to indicate the RAB hit ratio and RDB hit ratio of read access and overlay windows access, respectively. r_{OW} is higher than r_{RD} usually because overlay window access has a high spatial and temporal locality in LPDDR2-NVM. T_{pre} shows higher r_{RD} than that of the static optimum RDB configuration usually because it prefetches row buffer aggressively. This aggressive row buffer prefetches decrease r_{OW} of T_{pre}. M_{pre} reduces the total number of row buffer prefetches and increases the accuracy of prediction. Therefore, r_{RD} of M_{pre} are lower than that of T_{pre} in applications with a

sequential characteristic such as *bodytrack, raytrace, streamcluster,* and swaptions. r_{OW} of M_{pre} are higher than that of T_{pre} in all applications. Table 5.8 shows that $M_{pre} + OW$ increases r_{OW} while sacrificing little r_{RD}.

A RDB is allocated by a RDB miss or a row buffer prefetch. We define a ratio of row buffer prefetch, r_{PF}, as the number of RDB allocation caused by row buffer prefetches to the total number of RDB allocation. A row buffer prefetch is unnecessary if a prefetched RDB is not accessed until it is evicted. We define a good row buffer prefetch and a bad row buffer prefetch to evaluate a prediction of the heuristics. We consider it as a good row buffer prefetch if the prefetched RDB is accessed more than once before it is evicted and a bad row buffer prefetch if it is not accessed until it is evicted. We also define a ratio of good row buffer prefetch, $r_{G.PF}$, as the number of good row buffer prefetches to the number of total row buffer prefetches. The r_{PF} and $r_{G.PF}$ are good indicators that show the prediction accuracy of the heuristics.

Table 5.9 shows r_{PF} and $r_{G.PF}$ of T_{pre}, M_{pre}, and $M_{pre} + OW$. T_{pre} prefetches row buffer prefetch aggressively, in turn, r_{PF} of T_{pre} is higher than that of other heuristics. $r_{G.PF}$ of T_{pre} is also higher than other heuristics, but the number of bad row buffer prefetches is larger than other heuristics. This results in memory access time increase in some applications such as *blackscholes, canneal, fluidanimate,* and *x264*. Overlay window pinning of $M_{pre} + OW$ changes the victim selection. $M_{pre} + OW$ selects a RDB that contains read data as a victim instead of a RDB that contains overlay window when memory access pattern contains dense write accesses. It increases a RDB hit probability, and it results in increase of r_{PF} and $r_{G.PF}$.

We evaluate the total execution time reduction of the proposed heuristic. Figure 5.10 shows the comparison of total execution time of heuristics normalized to the static optimum RDB configuration on each application. APS takes on average 1.9% more time than the static optimum RDB configuration on average because it has lower RDB hit ratio. The total

Table 5.8: Comparison of the hit ratio.

Application	Static optimum		APS				T_{pre}		M_{pre}		$M_{pre}+OW$	
	r_{RD}	r_{OW}	$r_{RD.RAB}$	$r_{RD.RDB}$	$r_{OW.RAB}$	$r_{OW.RDB}$	r_{RD}	r_{OW}	r_{RD}	r_{OW}	r_{RD}	r_{OW}
blackscholes	16.8	96.3	49.7	14.2	77.7	22.3	29.0	85.9	27.1	96.2	27.1	96.6
bodytrack	35.7	81.8	57.2	35.4	61.8	38.2	63.2	78.5	61.1	81.1	61.1	82.0
canneal	0.6	74.9	36.1	0.9	97.4	2.6	0.5	63.0	0.7	75.0	0.7	76.5
dedup	30.6	82.5	65.1	29.0	69.2	30.8	39.8	77.9	42.2	82.0	42.2	82.4
facesim	56.6	68.0	62.2	35.9	63.4	36.6	73.7	72.8	74.3	74.6	74.2	76.7
ferret	58.2	82.6	54.0	43.5	55.4	44.6	62.0	84.2	69.8	85.0	69.7	85.3
fluidanimate	18.0	80.2	81.1	17.9	81.9	18.1	18.3	74.9	19.5	80.1	19.5	80.6
freqmine	52.5	78.2	56.7	41.0	58.0	42.0	56.7	81.9	61.5	83.5	61.5	83.9
raytrace	48.3	54.7	74.8	21.4	77.8	22.2	53.5	62.3	47.4	70.2	47.3	72.5
streamcluster	55.3	66.6	58.8	35.0	62.7	37.3	60.9	76.9	58.2	81.1	58.2	82.7
swaptions	67.4	71.5	55.8	43.1	56.4	43.6	88.1	74.8	86.9	75.2	86.9	76.7
vips	35.0	83.7	65.0	32.9	66.4	33.6	57.7	81.7	57.3	83.3	57.4	83.5
x264	33.5	82.0	56.7	34.0	62.6	37.4	48.7	76.7	50.7	81.3	50.7	82.3

Table 5.9: Comparison of the prefetch ratio and the good prefetch ratio.

Application	T_{pre}		M_{pre}		$M_{pre} + OW$	
	r_{PF}	$r_{G.PF}$	r_{PF}	$r_{G.PF}$	r_{PF}	$r_{G.PF}$
blackscholes	50.3	16.7	14.3	11.7	14.4	11.8
bodytrack	41.1	47.9	27.7	22.5	28.2	23.0
canneal	35.6	0.2	0.1	0.0	0.1	0.0
dedup	36.6	22.8	14.1	9.8	14.2	9.9
facesim	57.8	61.5	46.4	43.5	47.1	44.1
ferret	34.6	44.3	29.9	25.8	30.1	26.0
fluidanimate	32.9	2.6	5.1	1.0	5.2	1.1
freqmine	34.1	37.7	23.2	20.2	23.5	20.4
raytrace	51.6	41.2	24.5	20.4	24.6	20.5
streamcluster	50.3	47.4	32.7	27.3	33.1	27.5
swaptions	71.7	81.6	66.2	64.2	66.8	64.7
vips	39.1	41.1	23.8	19.0	23.9	19.1
x264	38.8	30.8	20.1	15.6	20.5	16.0

execution time of T_{pre} is on average 6.2% more than that of the static optimum RDB configuration. The memory access time of *canneal, dedup, fluidanimate,* and *x264* dominates the total execution time, and T_{pre} causes significant time overhead in these applications. The time reduction of M_{pre} is on average 1.9%. $M_{pre} + OW$ reduces on average 2.2% and up to 4.4% of total execution time. The proposed heuristic does not alter the CPU execution time but it shows significant total execution time reduction.

The cell program time is an important parameter of NVM. The dual operation of

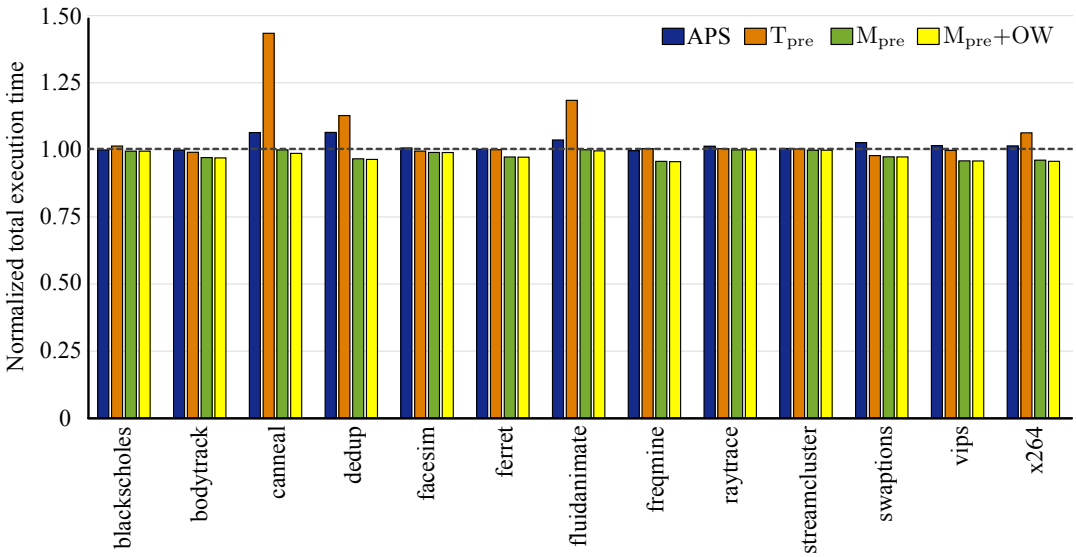
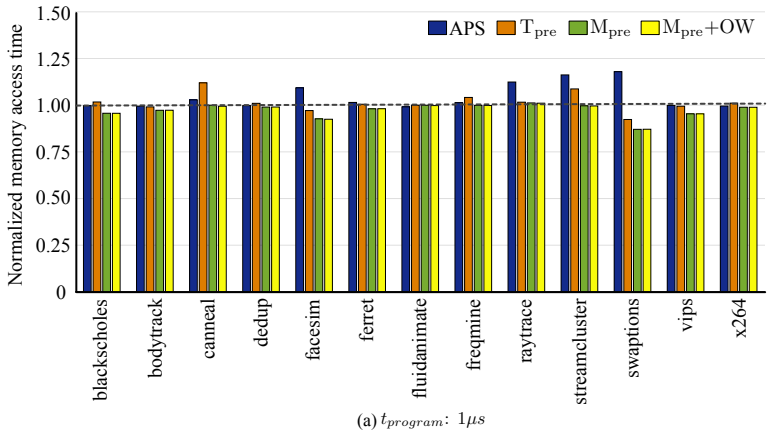


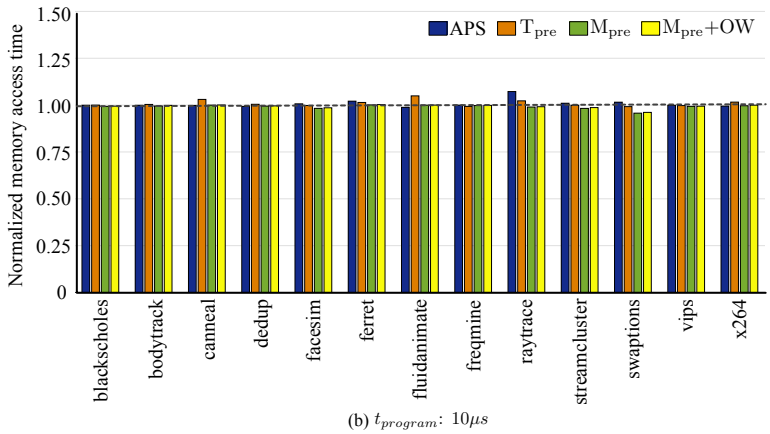
Figure 5.10: Comparison of total execution time normalized to the static optimum RDB configuration.

LPDDR2-NVM is able to hide cell program time in some cases, but it determines the performance of memory system usually. We use the cell program time of the optimistic research prototype in the evaluation [15]. We analyze the effect of cell program time on the memory access time reduction of the heuristics if the cell program time increases. Figure 5.11(a), 5.11(b), and 5.11(c) show the memory access time reduction when the cell program time is 1μs, 10μs, and 20μs, respectively. The memory access time reduction of the heuristic decreases as the cell program time increases because the cell program time dominates the memory access time. $M_{pre} + OW$ reduces up to 12.9%, 3.9%, and 2.9% of memory access time with 1μs, 10μs, and 20μs cell program time, respectively.

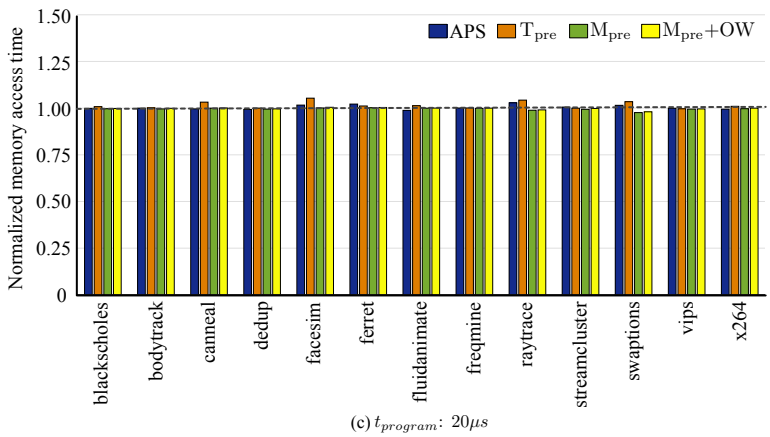
The characteristics of memory access pattern determines the memory access time reduction of the proposed heuristic. The number of cores in system causes the variation on the characteristics of memory access pattern. We analyze the potential of the proposed



(a) $t_{program}$: $1\mu s$



(b) $t_{program}$: $10\mu s$



(c) $t_{program}$: $20\mu s$

Figure 5.11: Memory access time varying on the program time. Memory access time is normalized to the static optimum configuration.

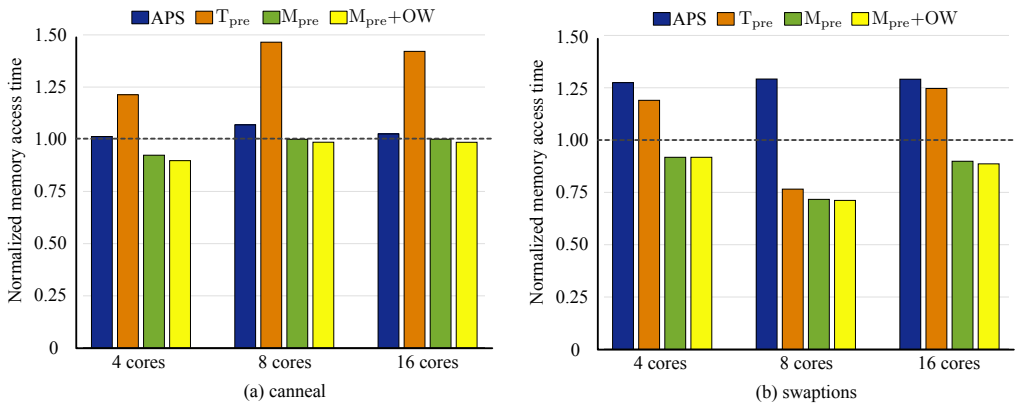


Figure 5.12: Sensitivity analysis of memory access time varying on the number of cores.

heuristics with respect to the variation of the number of cores, as shown in Figure 5.12. We select two representative applications – *canneal* and *swaptions* because those have the most random access pattern and sequential access pattern, respectively. The values of each heuristics are normalized to the memory access time of the static optimum RDB configuration. The number of cores variation is not related to the memory access time reduction of the heuristics, but it works as a different application, as shown in Figure 5.12.

We also analyze the relation between L2 cache size and the memory access time reduction of the heuristics. The simulated system uses 1 MB shared L2 cache. The number of memory access is reduced if the size of L2 cache increases, but the characteristics of memory access pattern does not change significantly. The proposed heuristic changes the logical RDB configuration according to the characteristics of memory access pattern, so the size of L2 cache does not affect the memory access time reduction of the proposed heuristic significantly.

Chapter 6

Conclusions

SRAM, DRAM, and NAND flash are main components of the typical memory systems, but these traditional memory devices have limitations such as volatility, low density, and high energy consumptions. Several innovative NVM technologies have been researched over the last few years to address the limitations of traditional memory devices. However, we found that many research papers do not correctly reflect NVMs' trends in the industry and overlook the standard interface, LPDDR2-NVM, although their contributions are significant.

This dissertation discusses the difference between the conventional DRAM and LPDDR2-NVM such as row buffer architecture, address mechanism, and asymmetric read/write operation. These differences require a different system-level performance optimization for the memory system using LPDDR2-NVM. We also review the previous work that adopting emerging NVMs to the typical memory systems.

For system-level optimization, we implement an LPDDR2-NVM platform that is made up of an LPDDR2-NVM prototype and a system-level simulator. The LPDDR2-NVM prototype verifies the operation of memory system with LPDDR2-NVM and provides parameters of LPDDR2-NVM to the system-level simulator. The system-level simulator is imple-

mented in SystemC, so it evaluates the effect of the system-level performance optimization cycle-accurately. There is no platform based on a real LPDDR2-NVM device. Therefore, the implemented platform provides an accurate information and insight to the researchers who work with NVMs.

The design space exploration shows the effect of row buffer management policy and configuration on the performance of memory system using LPDDR2-NVM. We compare the direct-mapped policy and fully-associate policy, and discuss the effect of replacement policy. Memory access time reduction varies on the RDB configuration and memory access pattern of applications. We analyze the effect of RDB configuration including other system parameters such as the number of cores and the size of shared L2 cache. From the results of the design space exploration, we concludes that RDB architecture is an key design factor to optimize the performance of memory system using LPDDR2-NVM.

Three-phase addressing mechanism and row buffer architecture of LPDDR2-NVM give us a way to optimize system performance. We propose a system-level method that improves the performance of memory system by skipping unnecessary addressing phase of three-phase addressing. We evaluate the effect of the proposed method by using LPDDR2-NVM platform. The proposed method reduce up to 41.8% memory access time where the cell program time reduces to 150ns. We also present the sensitivity analysis of memory access time varying on the cell program time. It enables us to predict the potential of the proposed method according to the cell program time variation.

The memory access pattern of application and the RDB configuration determines the performance of memory system with LPDDR2-NVM. The memory access pattern changes according to the application and/or time goes. We show the limitation of the static row buffer architecture and propose a system-level method that mimics reconfigurable row buffer architecture by managing row buffer proactively. The proposed method reduces the

memory access time on average 11.8% and up to 28.8% than the static optimum RDB configuration. We also analyze the relation between the memory access time reduction and the cell program time of NVM.

We discuss the performance of memory system that uses LPDDR2-NVM as main memory in this dissertation. LPDDR2-NVM has different addressing mechanism and row buffer architecture, so it requires different optimization method to improve the memory system performance. The long cell program time of NVM prevents the adoption of NVM in a memory system, but it is a promising DRAM replacement if the cell program time is reduced.

The following issues are remained for future research:

- Implementation of the LPDDR2-NVM controller that support a hybrid memory with DRAM and LPDDR2-NVM.
- Optimization of LPDDR2-NVM compatible memory device architecture including the size of partition.
- Cache write-back scheduling to reduce the latency of memory system using LPDDR2-NVM.

Bibliography

[1] M.-C. Yang, Y.-H. Chang, C.-W. Tsao, and P.-C. Huang, "New ERA: new efficient reliability-aware wear leveling for endurance enhancement of flash storage devices," in *Proceedings of the 50th Annual Design Automation Conference*, p. 163, 2013.

[2] Y. Xie, "Future memory and interconnect technologies," in *Procedings of the Design, Automation & Test in Europe Conference & Exhibition*, pp. 964–969, 2013.

[3] J. Cong, M. Ercegovac, M. Huang, S. Li, and B. Xiao, "Energy-efficient computing using adaptive table lookup based on nonvolatile memories," in *Proceedings of the 2013 IEEE International Symposium on Low Power Electronics and Design*, pp. 280–285, 2013.

[4] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, *et al.*, "Spin-transfer torque magnetic random access memory (stt-mram)," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 9, no. 2, p. 13, 2013.

[5] Y. Li, Y. Chen, and A. K. Jones, "A software approach for combating asymmetries of non-volatile memories," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pp. 191–196, 2012.

[6] JEDEC, "JESD209-F," 2013.

[7] "International Technology Roadmap for Semiconductors," 2012.

[8] C. Lam, "Cell Design Considerations for Phase Change Memory as a Universal Memory," in *Proceedings of the International Symposium on VLSI Technology, Systems and Applications*, pp. 132–133, 2008.

[9] R. Bez, "Chalcogenide PCM: a memory technology for next decade," in *Proceedings of the IEEE International Electron Devices Meeting*, pp. 5.1.1–5.1.4, 2009.

[10] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, pp. 439–447, July/September 2008.

[11] A. Bivens, P. Dube, M. Franceschini, J. Karidis, L. Lastras, and M. Tsao, "Architectural design for next generation heterogeneous memory systems," in *2010 IEEE International Memory Workshop (IMW)*, pp. 1–4, 2010.

[12] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th annual international symposium on computer architecture*, pp. 24–33, 2009.

[13] 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*, pp. 2–13, 2009.

[14] H. Chung, B. H. Jeong, B. Min, Y. Choi, B.-H. Cho, J. Shin, J. Kim, J. Sunwoo, J.-M. Park, Q. Wang, Y.-j. Lee, S. Cha, D. Kwon, S. Kim, S. Kim, Y. Rho, M.-H. Park, J. Kim, I. Song, S. Jun, J. Lee, K. Kim, K.-W. Lim, W.-R. Chung, C. Choi,

H. Cho, I. Shin, W. Jun, S. Hwang, K.-W. Song, K. Lee, S.-w. Chang, W.-Y. Cho, J.-H. Yoo, and Y.-H. Jun, "A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW," in *Proceedings of the 2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 500–502, 2011.

[15] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-j. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in *Proceedings of the 2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 46–48, 2012.

[16] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2010.

[17] C. Xu, D. Niu, Y. Zheng, S. Yu, and Y. Xie, "Reliability-aware cross-point resistive memory design," in *Proceedings of the 2014 Great Lakes Symposium on VLSI*, pp. 145–150, ACM, 2014.

[18] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proceedings of the IEEE*, vol. 98, pp. 2201–2227, December 2010.

[19] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam, "Phase-change random

access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4/5, pp. 465–479, 2008.

[20] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proceedings of the Design, Automation and Test in Europe*, pp. 136–141, 2010.

[21] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 329–338, ACM, 2011.

[22] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 243–252, ACM, 2012.

[23] J. Li, L. Shi, Q. Li, C. J. Xue, Y. Chen, Y. Xu, and W. Wang, "Low-energy volatile stt-ram cache design using cache-coherence-enabled adaptive refresh," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 1, p. 5, 2013.

[24] P. Mangalagiri, K. Sarpatwari, A. Yanamandra, V. Narayanan, Y. Xie, M. J. Irwin, and O. A. Karim, "A low-power phase change memory based hybrid cache architecture," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pp. 395–398, 2008.

[25] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proceedings of the 36th annual international symposium on Computer architecture*, pp. 34–45, 2009.

[26] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, pp. 14–23, 2009.

[27] W. Xu, J. Liu, and T. Zhang, "Data manipulation techniques to reduce phase change memory write energy," in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 237–242, 2009.

[28] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *Proceedings of the 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 2–13, 2009.

[29] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proceedings of the 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 347–357, 2009.

[30] 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. 131–141, 2010.

[31] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proceedings of the 37th annual International Symposium on Computer Architecture*, pp. 141–152, 2010.

[32] J. Hu, C. J. Xue, W.-C. Tseng, Q. Zhuge, and E. H. M. Sha, "Minimizing write activities to non-volatile memory via scheduling and recomputation," in *Proceedings of the 2010 IEEE 8th Symposium on Application Specific Processors*, pp. 101–106, 2010.

[33] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H. M. Sha, "Reducing write activities on non-volatile memories in embedded CMPs via data migration and recomputation," in *Proceedings of the 47th ACM/IEEE Design Automation Conference*, pp. 350–355, 2010.

[34] A. Seznec, "A Phase Change Memory as a Secure Main Memory," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 5–8, 2010.

[35] 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 fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, pp. 3–14, 2010.

[36] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 466–477, 2011.

[37] M. Joshi, W. Zhang, and T. Li, "Mercury: A fast and energy-efficient multi-level cell based Phase Change Memory system," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 345–356, 2011.

[38] J. Chen, Z. Winter, G. Venkataramani, and H. H. Huang, "rPRAM: Exploring Redundancy Techniques to Improve Lifetime of PCM-based Main Memory," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, pp. 201–202, 2011.

[39] C.-H. Chen, P.-C. Hsiu, T.-W. Kuo, C.-L. Yang, and C.-Y. M. Wang, "Age-based PCM wear leveling with nearly zero search cost," in *Proceedings of the 2012 49th ACM/EDAC/IEEE Design Automation Conference*, pp. 453–458, 2012.

[40] J. Chen, R. C. Chiang, H. H. Huang, and G. Venkataramani, "Energy-aware writes to non-volatile main memory," *ACM SIGOPS Operating Systems Review*, vol. 45, pp. 45–52, December 2012.

[41] A. Mirhoseini, M. Potkonjak, and F. Koushanfar, "Coding-based energy minimization for Phase Change Memory," in *Proceedings of the 49th ACM/IEEE Design Automation Conference*, pp. 68–76, 2012.

[42] 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 IEEE/ACM International Symposium on Microarchitecture*, pp. 14–23, 2009.

[43] 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 2010 IEEE 16th International Symposium on High Performance Computer Architecture*, pp. 1–11, 2010.

[44] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montano, and j. P. Karidis, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pp. 153–162, 2010.

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

[46] H. Park, S. Yoo, and S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," in *Proceedings of the 48th ACM/IEEE Design Automation Conference*, pp. 59–64, 2011.

[47] S. Bock, B. Childers, R. Melhem, D. Mosse, and Y. Zhang, "Analyzing the impact of useless write-backs on the endurance and energy consumption of PCM main memory," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 56–65, 2011.

[48] 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 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 478–489, 2011.

[49] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security Refresh: Protecting Phase-Change Memory against Malicious Wear Out," *IEEE Micro*, vol. 31, no. 1, pp. 119–127, 2011.

[50] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "DynRBLA: A high-performance and energy-efficient row buffer locality-aware caching policy for hybrid memories," *SAFARI Technical Report No. 2011-005*, 2011.

[51] 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 Annual International Symposium on Computer Architecture*, pp. 380–391, 2012.

[52] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "FPB: Fine-grained Power Budgeting to Improve Write Throughput of Multi-level Cell Phase Change Memory," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, 2012.

[53] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proceedings of the 2012 IEEE 18th International Symposium on High Performance Computer Architecture*, pp. 1–10, 2012.

[54] X. Zhang, L. Jang, Y. Zhang, C. Zhang, and J. Yang, "WoM-SET: Low power proactive-SET-based PCM write using WoM code," in *Proceedings of the 2013 ACM/IEEE international symposium on Low power electronics and design*, pp. 217–222, 2013.

[55] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture*, pp. 282–293, 2013.

[56] W. Zhang and T. Li, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures," in *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pp. 101–112, 2009.

[57] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proceedings of the 46th ACM/IEEE Design Automation Conference*, pp. 664–669, 2009.

[58] Y. Park, D.-J. Shin, S. K. Park, and K. H. Park, "Power-aware memory management for hybrid main memory," in *Proceedings of the 2nd international conference on next generation information technology*, pp. 82–85, 2011.

[59] L. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the international conference on supercomputing*, pp. 85–95, 2011.

[60] D.-J. Shin, S. K. Park, S. M. Kim, and K. H. Park, "Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory," in *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, pp. 395–402, 2012.

[61] T. Liu, Y. Zhao, C. J. Xue, and M. Li, "Power-Aware Variable Partitioning for DSPs With Hybrid PRAM and DRAM Main Memory," *IEEE Transactions on Signal Processing*, vol. 61, no. 14, pp. 3509–3520, 2013.

[62] Y. Park and K. H. Park, "High-Performance Scalable Flash File System Using Virtual Metadata Storage with Phase-Change RAM," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 321–334, 2011.

[63] G. S. Choi, I. Lee, M. Sung, and C. Im, "A hybrid SSD with PRAM and NAND Flash memory," *Microprocessors and Microsystems*, vol. 36, no. 3, pp. 257–266, 2012.

[64] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng, "A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems," in *Proceedings of the 8th ACM international conference on Embedded software*, pp. 31–40, 2008.

[65] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A Hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improve-

ment," in *Proceedings of the 2010 IEEE 16th International Symposium on High Performance Computer Architecture*, pp. 1–12, 2010.

[66] Y. Park, S.-H. Lim, C. Lee, and K. H. Park, "PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash," in *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1498–1503, 2008.

[67] H. G. Lee, "High-performance NAND and PRAM hybrid storage design for consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 56, pp. 112–118, Feb. 2010.

[68] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao, "PCM-FTL: A Write-Activity-Aware NAND Flash Memory Management Scheme for PCM-Based Embedded Systems," in *Proceedings of the 2011 IEEE 32nd Real-Time Systems Symposium*, pp. 357–366, 2011.

[69] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao, "A block-level flash memory management scheme for reducing write activities in PCM-based embedded systems," in *Proceedings of the Design, Automation and Test in Europe*, pp. 1447–1450, 2012.

[70] J. Guo, J. Yang, Y. Zhang, and Y. Chen, "Low cost power failure protection for MLC NAND flash storage systems with PRAM/DRAM hybrid buffer," in *Proceedings of the Design, Automation and Test in Europe*, pp. 859–864, 2013.

[71] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: A Protoype Phase Change Memory Storage Array," in *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, pp. 2–2, 2011.

[72] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[73] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, 2008.

[74] J. Park, D. Shin, N. Chang, and H. G. Lee, "Accelerating memory access with address phase skipping in lpddr2-nvm," *Journal of Semiconductor Technology and Science*, vol. 14, no. 6, pp. 741–749, 2014.

요약

일반적인 메모리 시스템에서는 캐쉬, 주메모리, 저장장치에 각각 SRAM, DRAM, NAND 플래시가 쓰여왔다. 하지만 이런 전통적인 메모리 소자는 휘발성, 낮은 집적성, 높은 누설 전력과 같은 단점들을 가지고 있다. 따라서, 비휘발성, 높은 집적성, 저전력의 특성을 가진 상변화 메모리, 스핀전달토크 메모리, 저항 변화 메모리와 같은 새로운 비휘발성 메모리 소자가 기존 메모리 소자의 대안으로 고려되고 있다. 새로운 비휘발성 메모리 소자의 다양한 장점으로 인해 많은 연구자들이 새로운 비휘발성 메모리 소자를 메모리 계층구조에 적용하는 방법에 대한 연구를 진행하고 있다.

새로운 비휘발성 메모리의 특성은 기존의 메모리와 다르기 때문에 최근 low power double data rate 2 non-volatile memory (LPDDR2-NVM) 표준이 비휘발성 메모리 소자를 연결하는 표준 인터페이스로 사용되고 있다. 하지만, 기존 연구의 대부분은 이러한 표준 인터페이스를 고려하지 않거나 간과하였다.

본 논문에서는 LPDDR2-NVM 기반 메모리 시스템의 성능을 최적화하기 위한 시스템 수준의 최적화 기법을 제안하였다. 이러한 목적을 위해 우선 LPDDR2-NVM 기반 메모리 시스템의 계수를 추출하기 위한 LPDDR2-NVM 프로토타입을 구현하였다. 또한, 현실적인 계수를 반영한 시스템 수준의 시뮬레이터를 제작하였다. 집중적인 실험를 통해 행 버퍼의 아키텍쳐가 LPDDR2-NVM 기반 메모리 시스템의 성능에 미치는 영향을 분석하였다. 이로부터 얻은 실마리를 통해 LPDDR2-NVM과 인터페이스하는 방식을

변형함으로써 성능을 향상시키는 시스템 수준의 기법을 제안하였다. 또한 정적 행 버퍼 구조의 한계점을 보이고 재구성형 행 버퍼 구조를 모방하는 시스템 수준의 기법을 제안하였다.

주요어: LPDDR2-NVM, 메모리 시스템, 성능 최적화

학번 2006-23166