

# 國立交通大學

電子工程學系 電子研究所

## 碩士論文

在多核心系統中考慮動態隨機存取記憶體讀/寫特性  
以降低功率消耗之排程機制

A Read-Write Aware DRAM Scheduling for Power  
Reduction in Multi-Core Systems

研究生：賴之彥

指導教授：周景揚 教授

中華民國一〇二年八月

在多核心系統中考慮動態隨機存取記憶體讀/寫特性  
以降低功率消耗之排程機制

**A Read-Write Aware DRAM Scheduling for Power  
Reduction in Multi-Core Systems**

研究生：賴之彥

Student: Chih-Yen Lai

指導教授：周景揚

Advisor: Jing-Yang Jou

國立交通大學

電子工程學系 電子研究所

碩士論文

A Thesis

Submitted to Department of Electronics Engineering and Institute of Electronics  
College of Electrical and Computer Engineering  
National Chiao Tung University  
In Partial Fulfillment of the Requirements  
for the Degree of  
Master of Science  
in  
Electronics Engineering

August 2013

Hsinchu, Taiwan, Republic of China

中華民國一〇二年八月

# 在多核心系統中考慮動態隨機存取記憶體讀/寫特性以降 低功率消耗之排程機制

學生：賴之彥

指導教授：周景揚 博士

國立交通大學

電子工程學系 電子研究所碩士班

## 摘要

近幾年來，隨著市場及業界對於高效能、低功耗系統的需求，功耗管理的重要性已日漸增加。在現今的多核心系統當中，動態隨機存取記憶體(DRAM)的功率消耗佔了整個系統的一大部分，因此吸引了許多人研究 DRAM 的功耗管理。而除了功率消耗之外，DRAM 同時還是目前多核心系統中的效能瓶頸，所以在設計 DRAM 功耗管理的方法時，必須格外地小心以避免大幅度地降低系統效能。目前關於 DRAM 功耗管理方法的研究中，有大量的研究是透過 DRAM 的排程機制來排序指令以降低功率消耗。在此基礎上，本篇論文提出考慮 DRAM 讀/寫特性的指令調節技術以及指令排程機制，在不影響系統效能的前提下，進一步降低 DRAM 功率消耗。根據實驗結果，本篇論文提出的機制平均能夠有效降低 75% 的 DRAM 功率消耗。若與現有的方法比較，本篇論文所提出的機制能夠在相同甚至是較少的系統效能損失之下，多降低 10% 的 DRAM 功率消耗。

# **A Read-Write Aware DRAM Scheduling for Power Reduction in Multi-Core Systems**

Student: Chih-Yen Lai

Advisor: Dr. Jing-Yang Jou

Department of Electronics Engineering  
Institute of Electronics  
National Chiao Tung University

## **ABSTRACT**

The demand of high performance and low power has increased the importance of power efficiency in multi-core systems. In modern multi-core architectures, DRAM has dominated the power consumption. Moreover, the performance of nowadays system is limited by the memory wall, which implies that a careless DRAM power management policy may harm the system performance dramatically. Among all the DRAM power management policies, reordering based DRAM scheduling has been widely studied to reduce the power. To further reduce the power while preserving the system performance, this thesis proposes the read-write aware throttling and the read-write reordering techniques. The proposed techniques effectively reduce 75% DRAM power on average. When compared to the existing work, the proposed techniques reduce 10% more power with comparable or less performance degradation on average.

# Acknowledgements

First and foremost, I would like to express my deepest appreciation to my advisor, Dr. Jing-Yang Jou for his patient guidance on my research. He also gave me many helpful suggestions about my career and motivated me to be a better person. I am sincerely grateful to my senior, Gung-Yu Pan for his kind guidance, direction and invaluable assistance. Discussions with him have been insightful, and I would never have been able to finish this thesis without his help. I would also like to thank my senior, Hsien-Kai Kuo for his useful suggestions. I appreciate all the members of EDA Lab, Bu-Ching Lin, Yung-Chun Lei, An-Che Cheng, and Chin-Fu Lu, for all your supports and for all the fun we have had together.

I would like to offer my special thanks to all my friends, including but not limited to: Chuan-Chia Huang, Kuan-Ting Chen, Kuan-Chang Wang, Yi-Jing Liu, Wen-Xuan Yu, for making my daily life joyful. Special thanks also go to all my teammates in NCTU EE volleyball team for filling up my life in the past six years, especially Ping-Yuan, Tsai, who has been a great help in tutoring me in almost every course I took. Thank all my lovely players in NCTU EE girl volleyball team, it was an honor coaching you. In particular, I would like to thank Chia-Ying Li for her assistance on many chores. I would also like to thank Samantha Koch, who gave me a lot of thoughts about my life.

Finally, I owe my deepest gratitude to my parents and my sister. Their endless support and love have always been encouraging. They have provided me a carefree environment so I can concentrate on my study. I am so lucky to be in such a wonderful family. At last but not least, thank my girlfriend, Chen-Huan Yen for being a huge part in my life. She stood by me through the good times and bad. I would not be who I am without her love.

Yoshi, Chih-Yen Lai    August, 2013



# Contents

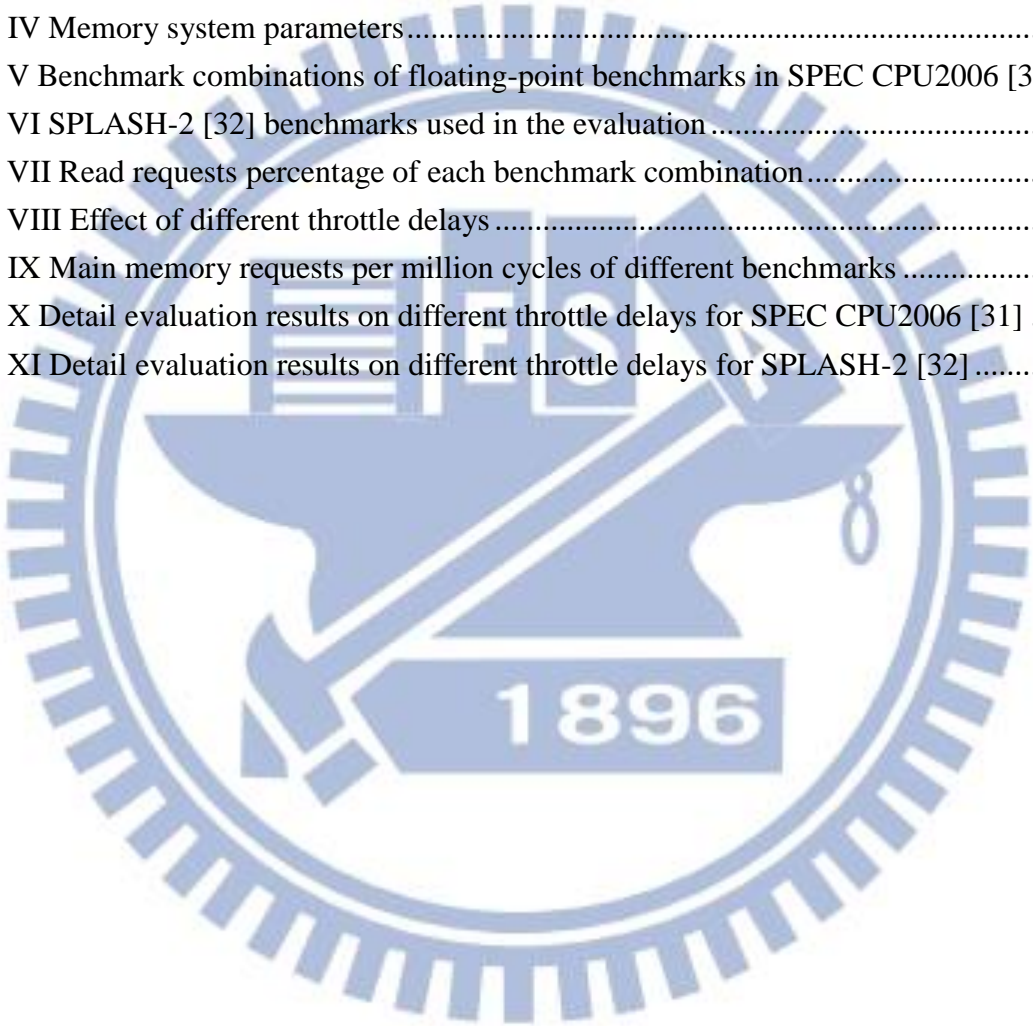
|   |     |
|---|-----|
| 摘 要.....                                    | I   |
| ABSTRACT.....                               | II  |
| Acknowledgements.....                       | III |
| Contents.....                               | IV  |
| List of Figures.....                        | V   |
| List of Tables.....                         | VI  |
| Chapter 1 Introduction.....                 | 1   |
| 1.1 DRAM Architecture.....                  | 1   |
| 1.2 Reducing the DRAM Power.....            | 5   |
| 1.3 Related Works and Motivation.....       | 7   |
| 1.4 Our Contributions.....                  | 11  |
| Chapter 2 Problem Description.....          | 12  |
| 2.1 System Model.....                       | 12  |
| 2.2 Problem Statement.....                  | 18  |
| Chapter 3 The Proposed Techniques.....      | 19  |
| 3.1 Overview.....                           | 19  |
| 3.2 Read-Write Aware Throttling.....        | 25  |
| 3.3 Rank Level Read-Write Reordering.....   | 29  |
| 3.4 An Example of The Proposed Policy.....  | 33  |
| Chapter 4 Experimental Results.....         | 36  |
| 4.1 Simulation Environment.....             | 36  |
| 4.2 Analysis on Different Techniques.....   | 40  |
| 4.3 Power and Performance Trade-Off.....    | 47  |
| Chapter 5 Conclusions and Future Works..... | 56  |
| References.....                             | 57  |

# List of Figures

|  |    |
|--|----|
| Fig. 1 The architecture of a DIMM, which includes ranks and banks, inside the DRAM. ....   | 2  |
| Fig. 2 Example of how a sequence of read commands to the same row is carried out inside a DRAM bank.....   | 3  |
| Fig. 3 Power mode transition delays. ....  | 6  |
| Fig. 4 System hierarchy diagram and the architecture of queues inside the memory controller.....   | 12 |
| Fig. 5 An example of how the blocked memory commands transfer from the RQ to the CQs when the throttle delay is reached in a throttling mechanism. ....                      | 15 |
| Fig. 6 Flow chart of a greedy memory controller, which employs the greedy power-down policy. ....  | 19 |
| Fig. 7 Flow chart of the memory controller proposed in the previous work [21]. ....  | 21 |
| Fig. 8 Flow chart of the memory controller employing the proposed techniques.....  | 23 |
| Fig. 9 An example of how commands blocked in the RQ are clustered into command sets....  | 26 |
| Fig. 10 An example of how commands in a given command set $S_1$ are combined into command groups and then reordered.....   | 30 |
| Fig. 11 An example of how the read-write aware throttling clusters commands in the RQ and determines which ranks should be turned on when the throttle delay is reached..... | 33 |
| Fig. 12 An example of how the commands in a given command set $S_1$ are reordered by the rank level read-write reordering and sent to the CQ.....                            | 34 |
| Fig. 13 Power and performance of different policies on different benchmark combinations. .   | 41 |
| Fig. 14 The background power, ACT/PRE power, and the read/write power consumptions of different techniques. ....   | 45 |
| Fig. 15 Average power and performance trade-off characteristics on SPEC CPU2006 [31]. ..   | 49 |
| Fig. 16 Power and performance trade-off characteristics for fp1.....   | 50 |
| Fig. 17 Power and performance trade-off characteristics for fp3.....   | 50 |
| Fig. 18 Average power and performance trade-off characteristics on SPLASH-2 [32]. ....   | 51 |

# List of Tables

|  |    |
|--|----|
| Table I Table of abbreviations .....   | 16 |
| Table II Table of notations .....  | 17 |
| Table III Configuration parameters of ARM Cortex A9 [30] .....                             | 37 |
| Table IV Memory system parameters .....  | 38 |
| Table V Benchmark combinations of floating-point benchmarks in SPEC CPU2006 [31] .....     | 38 |
| Table VI SPLASH-2 [32] benchmarks used in the evaluation .....                             | 39 |
| Table VII Read requests percentage of each benchmark combination .....                     | 43 |
| Table VIII Effect of different throttle delays .....                                       | 48 |
| Table IX Main memory requests per million cycles of different benchmarks .....             | 52 |
| Table X Detail evaluation results on different throttle delays for SPEC CPU2006 [31] ..... | 53 |
| Table XI Detail evaluation results on different throttle delays for SPLASH-2 [32] .....    | 54 |





# Chapter 1

## Introduction

In the latest multi-core systems, the increasing performance comes at the cost of the higher power consumption [1]. Within a multi-core architecture, it has been shown that main memory contributes up to 40% of the system power consumption [2][3][4]. For large scale systems such as datacenter, this fraction increases along the growing demand of memory capacity [5][6][7]. As the result, reducing the power consumption of main memory has become the main issue for chip designers.

Among existing memory circuits, *dynamic random access memory* (DRAM) is by far the mainstream of main memory used in modern multi-core systems. Hence, this thesis focuses on reducing the DRAM power consumption. To generalize the discussion, this thesis chooses to work on the widely used JEDEC standard DRAM architectures [8].

### 1.1 DRAM Architecture

Fig. 1 illustrates the architecture of a JEDEC standard DRAM. The JEDEC standard DRAM is composed of multiple *dual-in-line memory modules (DIMMs)*, in which multiple DRAM chips are put together to provide a wide data interface. Sets of DRAM chips lie in a DIMM. Each set of these DRAM chips is called a *rank*. Each rank can be further partitioned into *banks*, which spread across all DRAM chips within a rank. Each bank is organized as a two-dimensional array and its size is defined as  $rows \times columns$ . Only a single row of data can be accessed at any given time. Each bank has its own *row buffer* to store the row of data that is ready to be accessed [9]. Since the DRAM is a volatile memory, the data stored in the array need to be periodically refreshed to prevent data loss.

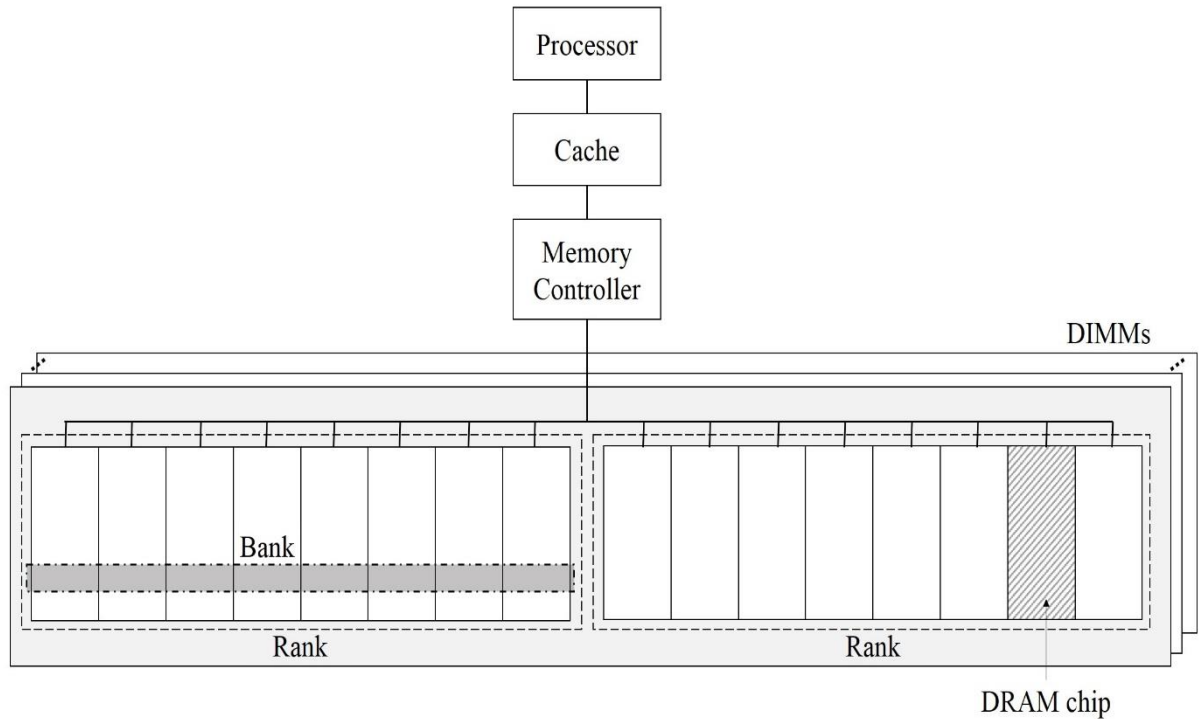


Fig. 1 The architecture of a DIMM, which includes ranks and banks, inside the DRAM.

The DRAM is operated by two types of commands, the internal commands and the request commands. The internal commands includes *refresh*, *activate (ACT)*, *precharge (PRE)*, *power-down* and *power-up*. All the internal commands are generated inside the DRAM. On the other hand, the request commands are the commands sent to the DRAM by the processor. A request command contains its access type and a target address indicating its target rank and target bank. Write request commands further contain the data to be written into the DIMMs. For simplicity, in the remainder of this thesis, the term *memory command* refers to the memory request command if not specified.

For a memory command to access a location in the DIMMs, several internal commands are triggered to carry out the read or write request. First, all the DRAM chips in the target rank of the memory command need to be activated. The DRAM generates an ACT command and sends it to the target bank of the memory command. The ACT command activates all the DRAM chips in the bank and reads out an entire row of data. This row of data is stored in the row buffer.

The data can either be read from or written to a column, which is indicated by the memory command, inside the fetched row. After the request completes, the DRAM send a PRE command to the open bank. The PRE command restores the data from the row buffer to the memory array inside the bank [10]. Fig. 2 shows an example of how a sequence of continuous read commands to the same row are carried out inside a DRAM bank.

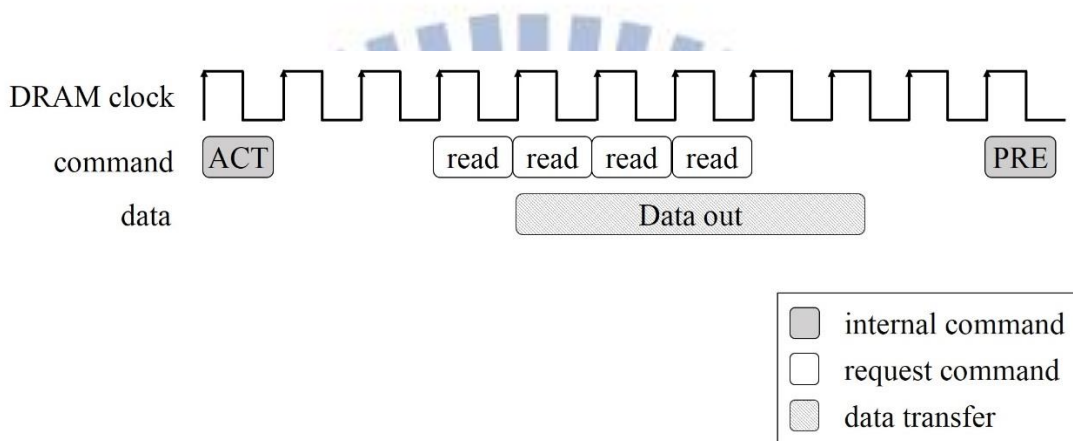
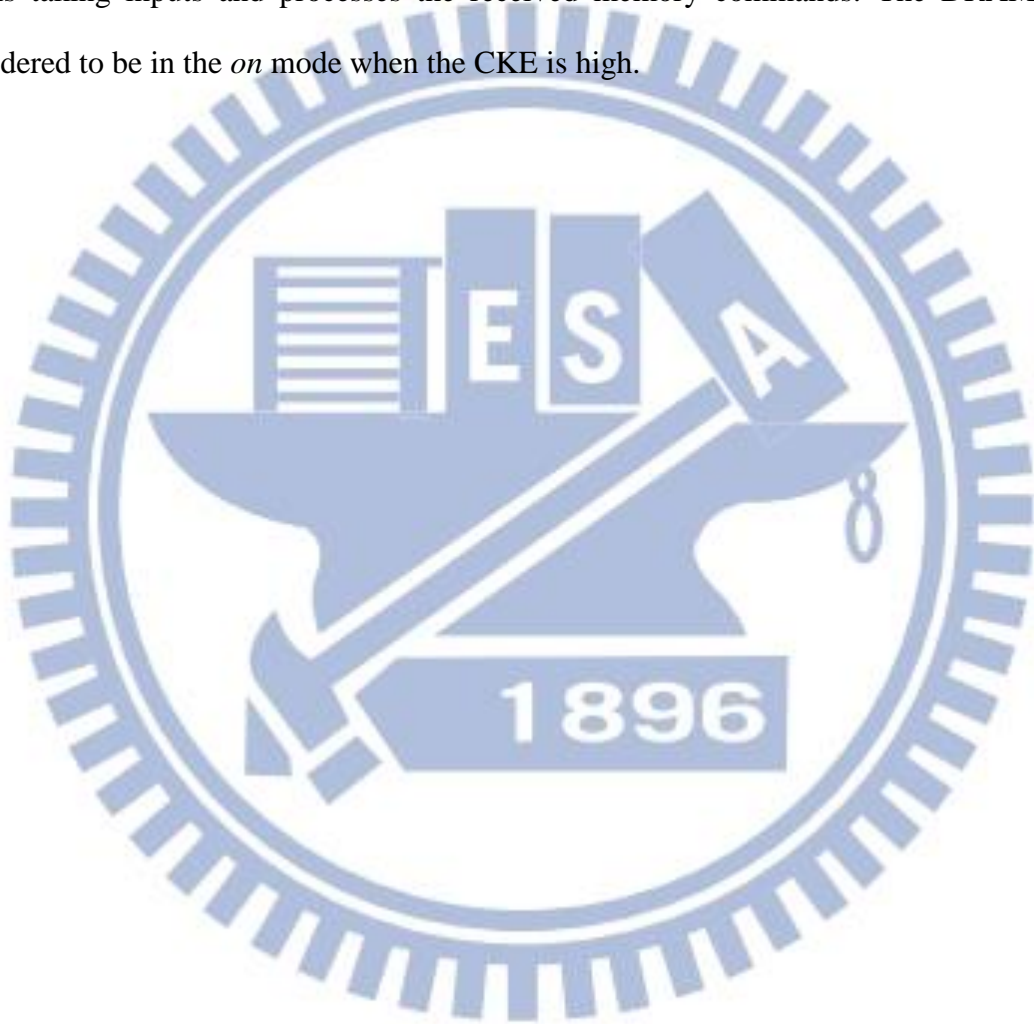


Fig. 2 Example of how a sequence of read commands to the same row is carried out inside a DRAM bank.

The DIMMs communicates with the processors through the *memory controller*, which is planted in the DRAM. The memory controller sits in between the last level cache and the DIMMs. It receives memory commands from the processor, and maps the addresses in the received commands to their target ranks and target banks inside the DIMMs. The memory controller then issues the received memory commands to the DIMMs. The memory controller is able to reorder the memory commands while preventing data hazards. The memory controller is also in charge of generating the internal commands such as ACT, PRE, and refresh. The memory controller can switch the power mode of a rank by generating power-down or power-up commands.

The master operation of a DRAM chip is controlled by the corresponding *clock enable (CKE)* signal. When CKE is low, all inputs, including DRAM clocks, are disabled [11]. If the

CKE for a DRAM chip is low, that DRAM chip is considered to be in the *off* mode. None of the request commands or the internal commands can be processed by a DRAM chip when it is in the off mode. The only exception is the refresh command. Even when a DRAM chip is in the off mode, it has to be periodically refreshed in order to retain data integrity in itself. The DRAM clocks start propagating through a DRAM chip when the CKE is risen to high. The DRAM chip begins taking inputs and processes the received memory commands. The DRAM chip is considered to be in the *on* mode when the CKE is high.





## 1.2 Reducing the DRAM Power

The power consumption of the DRAM consists of several different components. First of all, the DRAM continuously consumes a background power. The value of the background differs from the mode of the DRAM chip. When the DRAM chip is in the off mode, the background power is defined to be  $P_{PDN}$ . The background power is defined to be  $P_{ACT\_STBY}$  when the DRAM chip is in the on mode. It is natural that  $P_{ACT\_STBY}$  is larger than  $P_{PDN}$ ; for modern DRAM circuits,  $P_{PDN}$  is just 6.67%~15.38% of  $P_{ACT\_STBY}$  [10][11]. The periodic refresh operation consumes the refresh power ( $P_{REF}$ ). Finally, active power ( $P_{ACT}$ ), precharge power ( $P_{PRE}$ ), and the read or write power ( $P_{RD}$  or  $P_{WR}$ ) are average power needed for all memory accesses. To sum up, the power consumptions of the DRAM can be written as:

$$P_{off} = P_{PDN} + P_{REF} \quad (1)$$

$$P_{on} = P_{ACT\_STBY} + P_{REF} + P_{ACT} + P_{RD} + P_{WR} + P_{PRE}, \quad (2)$$

where  $P_{off}$  and  $P_{on}$  represent the power consumption of DRAM chips in the off mode and on mode respectively. The power consumption of DRAM chips in the off mode is the lowest power in which DRAM chips may keep the data integrity, and the off mode is therefore known as the *low power* mode.

Since the DRAM chips consume less power when they are in the low power mode, the DRAM chips that are idle should be turned off in order to reduce the DRAM power. Due to the fact that all the DRAM chips inside a rank need to be activated for a memory access, it is impossible to turn off a single DRAM chip alone. Therefore, a rank is the smallest set of DRAM chips that can be turned off. The JEDEC standard DRAM supports rank level power mode control and allows users to turn on or turn off each rank separately [8][12].

Although turning off idle ranks reduces the DRAM power, switching power mode of a rank takes a transition delay of time. Fig. 3 illustrates the power mode transition delays between two memory accesses, which can either be read or write (R/W). In Fig. 3, the memory accesses are depicted as rectangles, whereas  $t_{PDN}$  and  $t_{PUP}$  represent power-down transition delay and power-up transition delay respectively. The delays of switching power mode of ranks bring extra latencies to the system performance. Therefore, one of the main goals for designing DRAM power reduction techniques is to design a policy that determines when to turn on and off DRAM ranks. An immature policy leads to loss of power saving or serious degradation to the system performance.

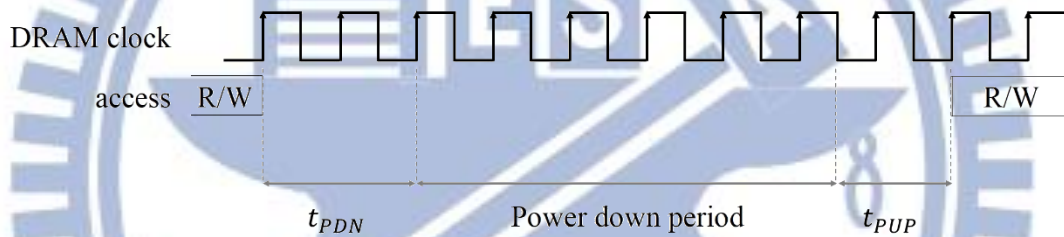


Fig. 3 Power mode transition delays.

## 1.3 Related Works and Motivation

Many different types of approaches toward DRAM power reduction have been studied in recent years. These approaches include designing a hybrid main memory, re-designing the physical structure of the DRAM, adding hardware component to the DRAM and designing power management policy for the memory controller in the modern DRAM.

Designing a hybrid main memory aims to reduce the DRAM power consumption by cutting down the refresh energy consumed by the DRAM. Phase change random access memory (PCRAM) is used in the hybrid memory as a large background main memory since it consumes low standby power. The DRAM serves as a cache above PCRAM [13][14]. The data in DRAM decay over time and are written back to the PCRAM if they are dirty. The DRAM refresh energy is thus reduced. Besides integrating PCRAM with DRAM, some proposed to use the cached DRAM, which adds a cache directly into the memory device [15]. Adding cache to the DRAM reduces the access to DRAM chips and thus increases the idle period. Therefore, the DRAM chips can be put to the low power mode to save more power.

Re-designing the physical structure of DRAM targets to improve the granularity of DRAM power mode control, which increases the potential of turning off idle DRAM chips. Some proposed to separate ranks into mini-ranks by adding mini-rank buffers inside each rank [16]; others proposed to change the arrangement of arrays in each bank [2]. These approaches allow the memory controller to switch the power mode of a set of DRAM chips, which the number of chip is smaller than that contained in a rank. It is more likely for DRAM chips in a smaller chip set to be all idle. Therefore, the memory controller has more chance to turn off idle chips and thus reduces the DRAM power consumption.

Both designing hybrid main memories and re-designing the physical structure of the DRAM have the potential to achieve good performance. Nevertheless, they both require big modifications to the modern DRAM architecture. Since this thesis aims to design power



reduction techniques for DRAMs that are commonly used in nowadays multi-core systems, these approaches are not suitable.

A different type of approaches reduce the DRAM power by adding some extra hardware to the existing DRAM circuit and extend its capability. For example, retention-aware intelligent DRAM refresh mechanism identifies and skips unnecessary refreshes for the DRAM [17]. By only refreshing necessary rows in DRAM banks, the refresh power is cut down. Another example is automatic data migration, which migrates data from ranks to ranks and tries to make the memory access concentrate on certain ranks [18]. This creates more empty ranks, which do not need to be periodically refreshed and can be completely shut off for a period of time. However, these approaches require a large hardware overhead to implement.

In order to reduce the DRAM power consumption without a big modification to the modern DRAM circuits, many approaches design power management policies for the memory controllers. The policies can be categorized into power-down policies, which determine when to turn an idle rank off; scheduling policies, which schedule the commands in the memory controller; throttling-based policies, which block command in the buffer to prolong the idle period of a rank.

An intuitive power-down policy is the *time-out power-down* policy [19]. The time-out power-down policy turns off a rank once it is idle for a pre-defined, fixed period of cycles, regardless of the upcoming command pattern. This results in inflexible power mode transitions, which may turn off the DRAM ranks even during a short idle period or waste active standby power waiting for the timer to expire. Therefore, the time-out power-down policy does not guarantee an acceptable power reduction and may harm the system performance dramatically [20]. Besides the time-out power-down policy, *queue-aware power-down* policy is proposed [21]. In the queue-aware power-down policy, if a rank is idle, commands in the memory controller are checked to see if any of the pending commands is destined for the idle rank. If there is no pending commands heading to the idle rank, it is turned off. The aggressive



bandwidth-neutral strategy proposed in [22] is a policy derived from the queue-aware power-down policy with an additional snooping mechanism that turns the idle ranks back on upon receiving a memory command in the memory controller. The queue-aware power-down policy and the aggressive bandwidth-neutral strategy do not affect the system performance as much as the time-out power-down policy does, but their effect on reduction DRAM power is limited because they do not maximize the ability of the memory controller.

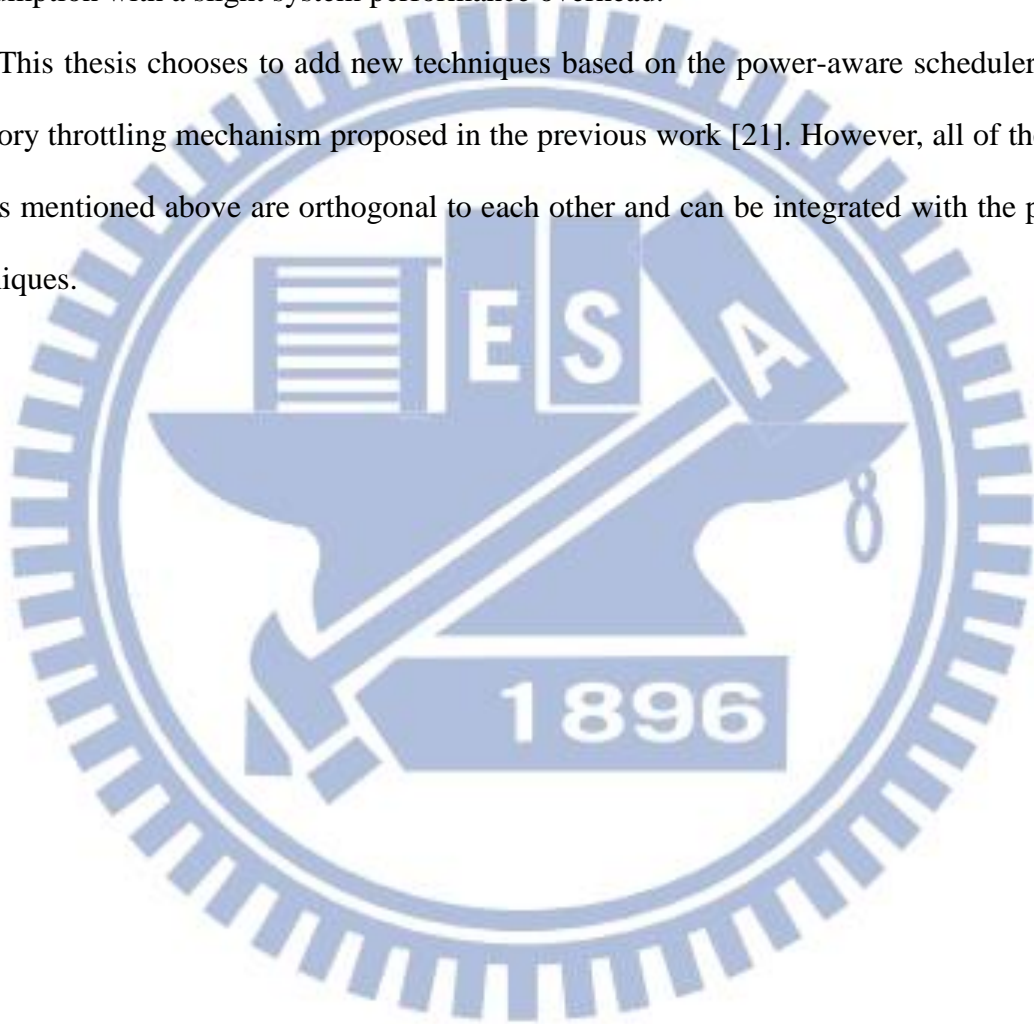
The scheduling policies are also known as schedulers. Some schedulers stress on scheduling both request commands and internal commands [9][23][24]. These schedulers require extra hardware to allow the memory controller arranges all the commands more sophisticatedly to reduce the DRAM power. Other schedulers focus on scheduling only the request commands. Since the memory controller reorders request commands while preventing data hazards even when no power management policy is employed, scheduling request commands does not add extra hardware to the memory controller. Among this type of schedulers, the *power-aware memory scheduler* is proposed in [21]. The power-aware memory scheduler clusters memory commands destined for the same rank together. While the memory concentrates on accessing one rank, other ranks can be switched to the low power mode to save power. Since only a single rank is activated at a time, other ranks stay in the low power mode for longer periods.

The throttling-based policies restricts memory access commands to be issued to the DIMMs to reduce the DRAM power [21][25]. The *memory throttling* mechanism proposed in [21] increases the power saving by blocking memory commands in the memory controller and force the DRAM chips to stay idle. No command is issued by the memory controller before it has blocked commands for a fixed period of cycles. This fixed period of cycles is called the *throttle delay* ( $t_{TD}$ ).

With the queue-aware power-down policy, power-aware scheduler and the memory throttling mechanism, the previous work [21] achieves a power saving closed to time-out

power-down policy with a moderate system performance degradation. However, the previous work [21] does not take into consideration that read requests are more critical for the system performance than write requests [26]. In modern multi-core systems, a higher write latency can be tolerated using buffers [27]. Therefore, slowing down the write accesses does not impact the system performance much. Utilizing this fact, one can further reduce DRAM power consumption with a slight system performance overhead.

This thesis chooses to add new techniques based on the power-aware scheduler and the memory throttling mechanism proposed in the previous work [21]. However, all of the related works mentioned above are orthogonal to each other and can be integrated with the proposed techniques.



## 1.4 Our Contributions

To reduce DRAM power consumption, this thesis proposes the read-write aware DRAM scheduling, which utilizes the different criticalities between read accesses and write accesses. The DRAM scheduling mentioned in the remainder of this thesis refers to reordering the sequence of all the memory commands inside the memory controller. In this thesis, the internal DRAM commands such as ACT and PRE...etc. remain their default order and are not taken into consideration in the proposed techniques.

This thesis proposes two techniques, the *read-write aware throttling* mechanism and the *rank level read-write reordering* technique. The read-write aware throttling mechanism effectively cuts down DRAM power consumption. The rank level read-write reordering is employed to significantly reduce the system performance degradation caused by DRAM power management while maintaining the power saving. Our work reduces 75.34% of the DRAM power from the DRAM with no power management. When compared to the existing work, the proposed techniques reduce 10% more DRAM power with less performance degradation. Moreover, by evaluating and comparing with an oracle policy, the experimental results have shown that our work can reduce more power at the expense of a slight system performance degradation.

The remainder of this thesis is organized as follows. The next chapter describes the problem formulation. Chapter 3 places the proposed techniques. The experiment results are presented in Chapter 4. Finally, this thesis is concluded in Chapter 5.

# Chapter 2

## Problem Description

### 2.1 System Model

As shown in Fig. 4, the system has one or more processor cores. These cores are connected to a multi-level cache and are equipped with write buffers. A JEDEC standard DRAM is used as a shared main memory for all cores. The DRAM main memory is connected to the last level cache. The DRAM communicates with the processors through a memory controller, which lies between the last level cache and the DRAM circuit. The memory controller receives memory commands from the processors and issues these commands to the DIMMs inside the DRAM. Each memory command contains information including its target address and access type. The memory controller keeps track of the time it receives each command.

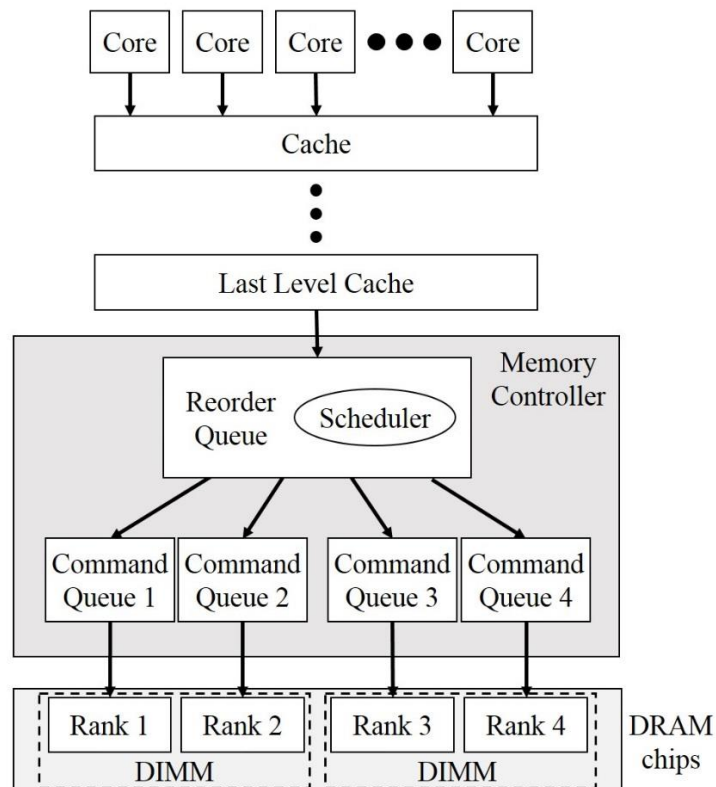


Fig. 4 System hierarchy diagram and the architecture of queues inside the memory controller.



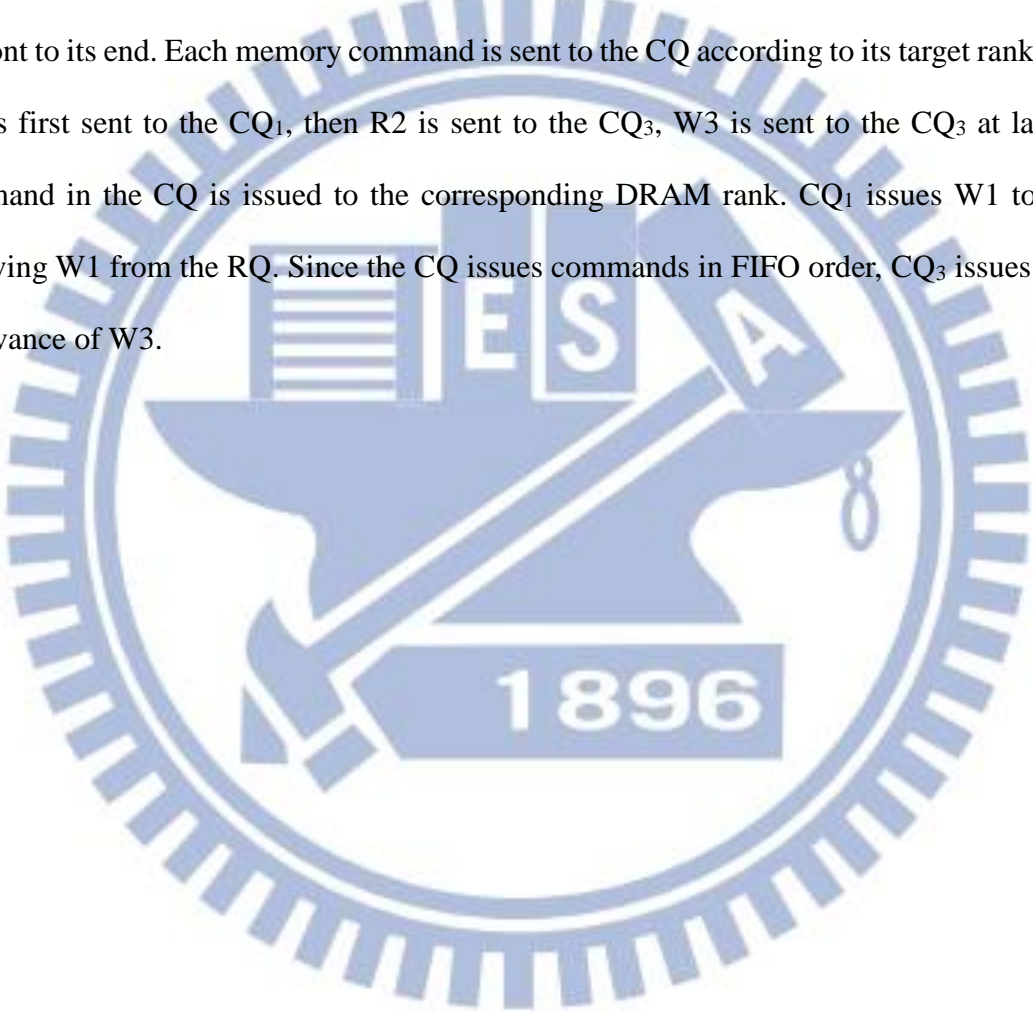
Within the memory controller, there are two queues: the *reorder queue (RQ)* and the *command queues (CQs)*. Memory access commands from the last level cache are first stored in the RQ. These commands are mapped to certain DRAM ranks and banks according to their target addresses. A scheduler inside the RQ is able to reorder the commands in the RQ while keeping the data hazard-free. After mapping and reordering, the memory controller sends the memory commands from the RQ to the CQs one at a cycle. Each CQ corresponds to a certain rank. Therefore, for a DRAM with  $n$  ranks, there are  $n$  CQs inside the memory controller. Memory commands destined for rank 1, rank 2... rank  $n$  are sent to  $CQ_1, CQ_2 \dots$  and  $CQ_n$ , respectively. A CQ handles all kinds of commands destined for its corresponding rank. These commands not only include access commands from the RQ, but also include other internal commands such as ACT, PRE, refresh, power-up, and power-down commands generated by the memory controller. The CQs do not change the order of the memory access commands since they are already reordered by the scheduler inside the RQ. Hence, the CQs issue the commands to the DRAM ranks in a first-in-first-out (FIFO) order.

Normally, the memory controller sends the commands from the RQ to the CQs whenever there are commands in the RQ. When a throttling mechanism is employed, instead of sending the memory access commands from the RQ to the CQs whenever the RQ is not empty, the memory controller blocks commands in the RQ for  $t_{TD}$ . No command is sent to the CQs before the throttle delay is reached. When the throttle delay is reached, the memory controller starts sending the blocked commands to the CQs one at a cycle until the RQ is empty. The memory controller repeatedly blocks and releases the commands to realize the throttling mechanism.

Fig. 5 gives an example of how the memory commands are transferred from the RQ to the CQs when the throttling mechanism is employed. In Fig. 5, each rectangle represents a memory command. The access type of each command is denoted by R (read) or W (write), followed by an index number and the target rank of the command. The index numbers are assigned to each memory command according to the time they entered the RQ. The command that enters the RQ

earlier is represented by a smaller index number. For the example shown in Fig. 5, the DRAM is assumed to have four ranks, which are denoted as  $r_1$ ,  $r_2$ ,  $r_3$ , and  $r_4$ . Correspondingly, there are four CQs in the memory controller. Suppose that the throttle delay is reached at cycle  $k$ . For simplicity, we denote the  $i^{\text{th}}$  rank as  $r_i$  in Fig. 5 as well as in the remainder of this thesis.

As shown in Fig. 5, three commands W1, R2, and W3 are blocked in the RQ during cycle  $k - t_{TD}$  to cycle  $k$ . When the throttle delay is reached, the RQ starts sending commands from its front to its end. Each memory command is sent to the CQ according to its target rank. Hence, W1 is first sent to the CQ<sub>1</sub>, then R2 is sent to the CQ<sub>3</sub>, W3 is sent to the CQ<sub>3</sub> at last. Each command in the CQ is issued to the corresponding DRAM rank. CQ<sub>1</sub> issues W1 to  $r_1$  after receiving W1 from the RQ. Since the CQ issues commands in FIFO order, CQ<sub>3</sub> issues R2 to  $r_3$  in advance of W3.



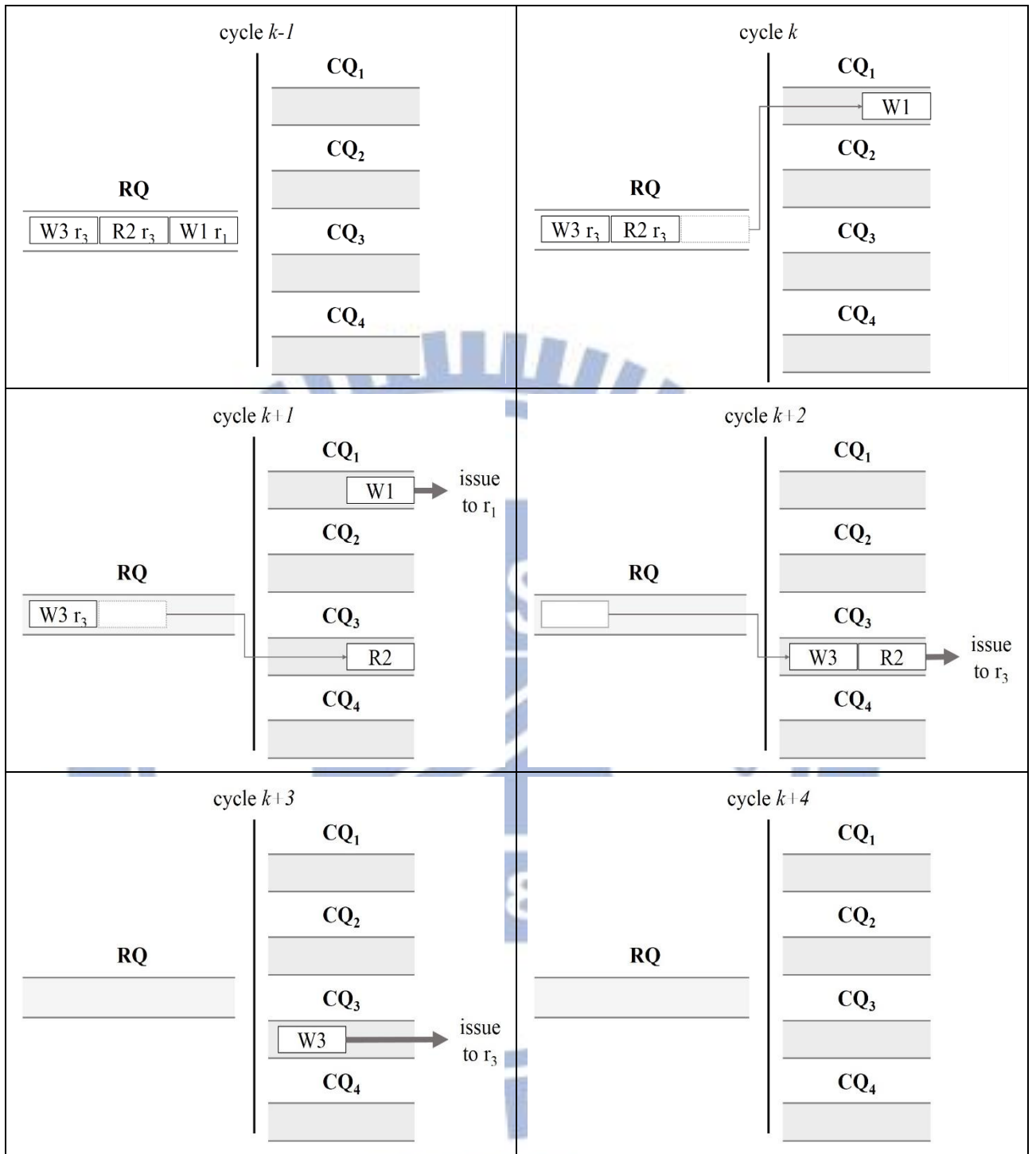


Fig. 5 An example of how the blocked memory commands transfer from the RQ to the CQs when the throttle delay is reached in a throttling mechanism.

The DRAM supports rank level power-mode control. That is, each rank has two different power mode, the active mode and the low power mode. A rank has to be turned on to the active mode before it is able to process the received commands. When a rank is idle, it can be turned off to the low power mode by the memory controller. The memory controller puts power-up and power-down commands into the CQ to switch the power mode of its corresponding DRAM rank. Switching the power mode of a rank is at the cost of transition delays.

The abbreviations and notations used throughout this thesis are listed in the following Table I and Table II.

Table I  
Table of abbreviations

| <i>Abbreviation</i>    | <i>Definition</i>                           |
|------------------------|---|
| DRAM                   | dynamic random access memory                |
| DIMM                   | dual-in-line memory module                  |
| ACT                    | activate                                    |
| PRE                    | precharge                                   |
| CKE                    | clock enable                                |
| PCRAM                  | phase change random access memory           |
| RQ                     | reorder queue                               |
| CQ                     | command queue                               |
| CQ <sub><i>i</i></sub> | the command queue assigned to rank <i>i</i> |
| FIFO                   | first-in-first-out                          |
| r <sub><i>i</i></sub>  | rank <i>i</i>                               |
| RAW                    | read after write                            |
| MIPS                   | million instructions per second             |



Table II  
Table of notations

| <i>Notation</i> | <i>Definition</i>   |
|-----------------|---|
| $P_{PDN}$       | background power of a DRAM chip in the off mode             |
| $P_{ACT\_STBY}$ | background power of a DRAM chip in the on mode              |
| $P_{REF}$       | power consumption of refresh operation                      |
| $P_{ACT}$       | power consumption of an ACT command                         |
| $P_{PRE}$       | power consumption of a precharge command                    |
| $P_{RD}$        | average power consumption of read accesses                  |
| $P_{WR}$        | average power consumption of write accesses                 |
| $P_{off}$       | total power consumption of a DRAM chip in the off mode      |
| $P_{on}$        | total power consumption of a DRAM chip in the on mode       |
| $t_{PDN}$       | power down transition delay                                 |
| $t_{PUP}$       | power up transition delay                                   |
| $t_{TD}$        | throttle delay  |
| $n$             | number of ranks in the DRAM                                 |
| $C_{RQ}$        | number of commands in the RQ                                |
| $S_i$           | the $i^{\text{th}}$ command set                             |
| $C_{Si}$        | number of commands in the $i^{\text{th}}$ command set       |
| $R_i$           | number of read commands in the $i^{\text{th}}$ command set  |
| $W_i$           | number of write commands in the $i^{\text{th}}$ command set |

## 2.2 Problem Statement

With the system model described in the previous section, the goal of this thesis is to find a delicate DRAM scheduling scheme that reduces the DRAM power with small system performance degradation for the memory controller. The scheduling scheme includes a throttling mechanism, which controls when the RQ starts sending commands to the CQs. The scheme also contains a scheduling policy for the scheduler inside the RQ, which is able to reorder the sequence of memory request commands in the RQ. The internal commands such as ACT and PRE are not taken into consideration by the scheduler inside the RQ since they are generated and put to the CQs directly by the memory controller. The scheduling policy should guarantee that the reordered sequence of memory commands are hazard-free. Finally, the scheme is in charge of determining when to turn on and off the DRAM ranks.

By using the throttling mechanism, reordering the memory commands and controlling the power mode of ranks, the goal of the proposed scheme is to reduce the power consumption of the DRAM with minor system performance degradation.

# Chapter 3

## The Proposed Techniques

### 3.1 Overview

The proposed DRAM power reduction techniques address on lowering DRAM power consumption with slight system performance degradation. Since the proposed techniques are based on the existing policies, this section starts off with the basic and shows the flow chart of the greedy memory controller in Fig. 6.

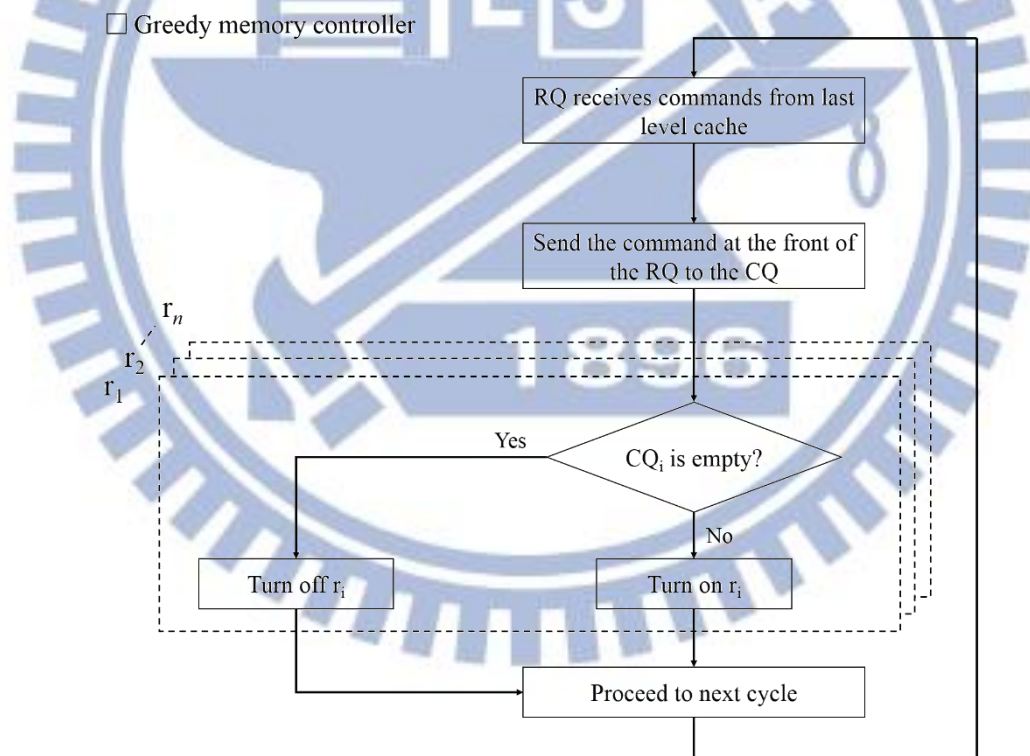


Fig. 6 Flow chart of a greedy memory controller, which employs the greedy power-down policy.

The greedy memory controller employs the greedy power-down policy, which turns off a DRAM rank whenever it is idle. In the greedy power-down policy, an idle rank is turned off even when there are pending commands destined for it in the RQ. The greedy memory controller does not employ the throttling mechanism. Therefore, whenever the RQ is not empty, a single memory command is sent from the RQ to the corresponding CQ every cycle.

As shown in Fig. 6, at any given cycle, the memory commands from the last level cache are pushed to the end of the RQ. The memory controller checks the state of each rank and turns off the ranks that are idle. After turning off idle ranks, the memory controller sends the command at the front of the RQ to its corresponding CQ. Finally, the memory controller checks each CQ to see if they are empty. If CQ<sub>i</sub> is not empty, the memory controller turns on  $r_i$  and issues the commands to it.

Since the greedy memory controller turns off an idle rank regardless of the upcoming memory commands in the RQ, the idle rank is turned off even during a short idle period. This results in frequent power mode transition and leads to dramatic system performance degradation caused by the power mode transition delays [20].

To improve the power reduction and the system performance of the greedy memory controller, the previous work [21] adds the throttling mechanism, the queue-aware power-down policy, and the power-aware memory scheduler to the greedy memory controller. The flow chart of the previous work [21] is shown in Fig. 7.



- Greedy memory controller
- ▒ Previous work [21]

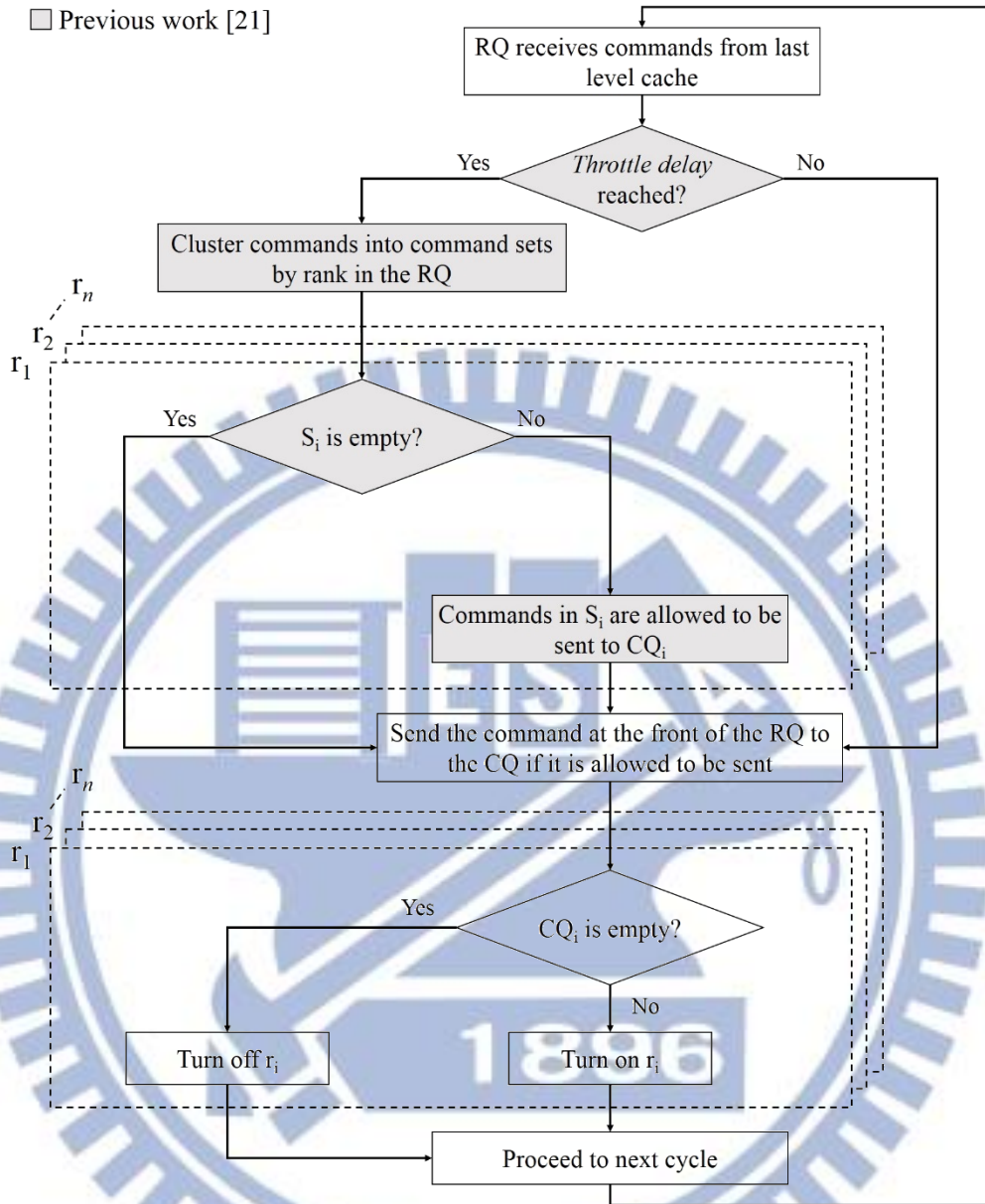


Fig. 7 Flow chart of the memory controller proposed in the previous work [21].

The light-gray rectangles and decision boxes in Fig. 7 are power reduction techniques added by the previous work [21]. The throttling mechanism, which is shown as the first light-gray decision box in the flow chart, blocks memory commands after they are pushed to the RQ until the throttle delay is reached. The blocked commands are not allowed to be sent to the CQ. When the throttle delay is reached, the memory controller clusters the blocked commands into *command sets* by their target ranks. Commands destined for  $r_i$  are clustered into command set

$S_i$ . The set of commands destined for the same rank as the command that first entered the RQ are moved to the front of the RQ, and so on. The memory controller then checks each command set, if there is command in  $S_i$ , all the commands in  $S_i$  are allowed to be sent to the CQ. The memory controller then sends command at the front of the RQ to the corresponding CQ. At the end of each cycle, the memory checks each CQ to see if there is commands in it. If  $CQ_i$  is empty, the memory controller sends a power-down command to  $r_i$ , which turns  $r_i$  off.

To sum up, the techniques proposed in the previous work [21] improves both power reduction and system performance from the greedy memory controller. The power-aware memory scheduler in the previous work clusters commands according to their target ranks. This forces the DRAM to concentrate on accessing a certain rank for a period of time, allowing other ranks to be turned off. The queue-aware power-down policy checks the upcoming commands in the CQ before turning off a rank, which prevents a rank to be turned off during a short idle period. When a rank is turned off, the throttling mechanism assures that the rank stays in the low power mode for a long period of time.

However, the previous work [21] does not maximize the capability of the memory controller. Moreover, it does not take into consideration that the read requests are more critical to the system performance than the write requests. Utilizing this fact, this thesis proposes the read-write aware throttling and the rank level read-write reordering techniques to modify the previous work [21]. The flow chart of the memory controller that employs the proposed techniques is shown in Fig. 8.

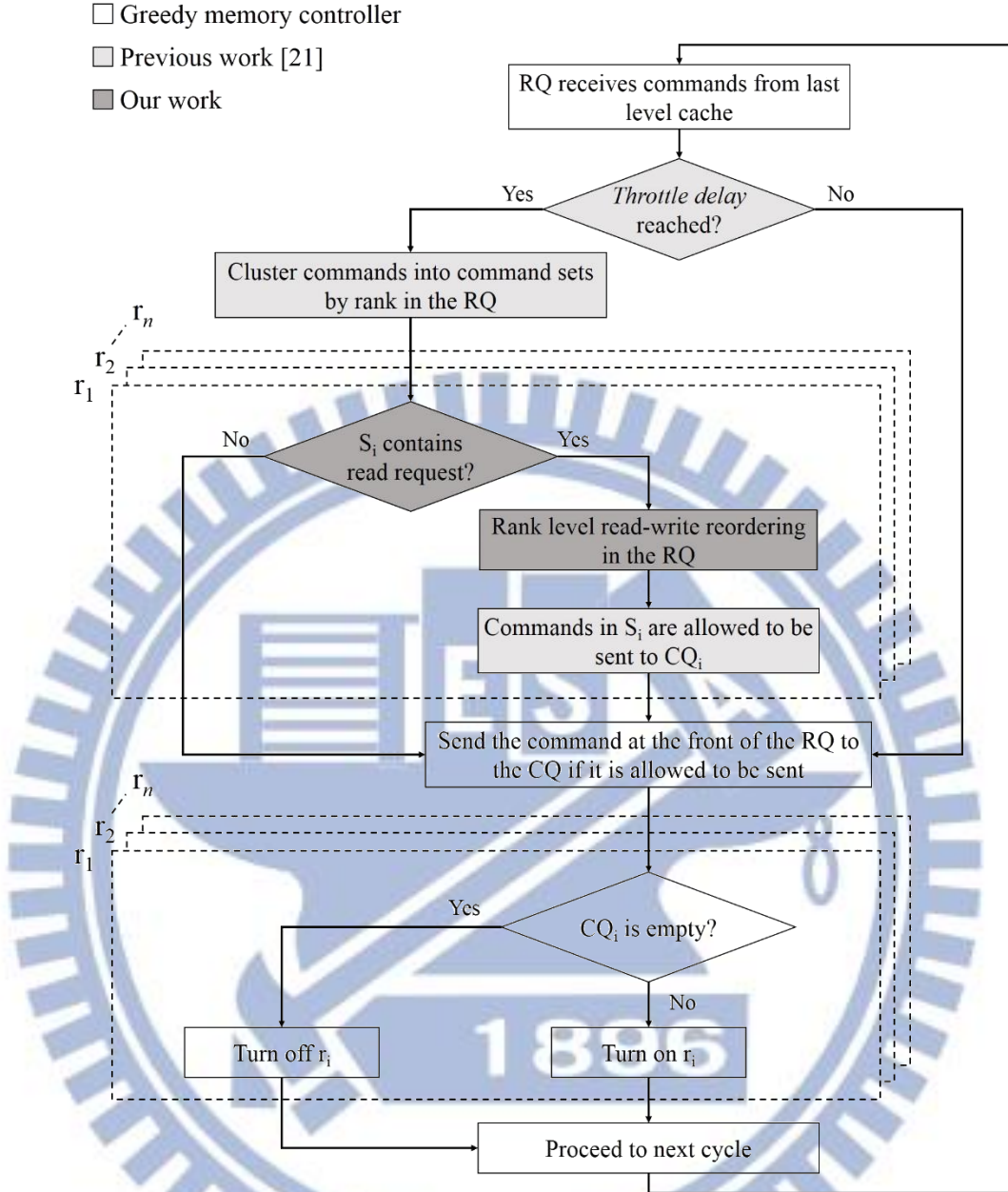


Fig. 8 Flow chart of the memory controller employing the proposed techniques.

Based on the previous work [21], the dark-gray rectangle and box in Fig. 8 are the techniques proposed in this thesis, where the decision box is the *read-write aware throttling* and the rectangle is the *rank level read-write reordering*.

The read-write aware throttling mechanism, which is depicted by the dark-gray decision box, checks for the existence of read commands in each command set. Instead of allowing all the nonempty command sets to be sent to the CQs, only the command sets containing read

requests are allowed to be sent to the CQs and only their target ranks are turned on. The other ranks, including ranks with write requests pending in the RQ, stay in the low power mode for another throttle delay to reduce the DRAM power consumption.

The dark-gray rectangle shows that the command sets are reordered by the rank level read-write reordering before they are sent to the CQs. The read requests in each command set get higher priorities than the write requests. The commands with higher priorities enter the CQs earlier. Since the CQ issues commands to the DIMM in a FIFO order, the read requests reach their target rank as soon as possible. This makes read requests, which are critical to system performance, to be served by the DIMMs earlier and thus the system performance degradation caused by the throttling mechanism is relieved.

For simplicity, the following notations are used in this thesis. Suppose that there are  $n$  ranks in the DRAM. There is one RQ and  $n$  CQs,  $CQ_1, CQ_2 \dots$  and  $CQ_n$ , inside the memory controller. The number of commands blocked inside the RQ is denoted as  $C_{RQ}$ . Inside the RQ, commands destined for  $r_1, r_2 \dots$  and  $r_n$  are clustered into command sets  $S_1, S_2, \dots$ , and  $S_n$  respectively. The notation  $C_{Si}$  represents the number of commands in the command set  $S_i$ . For each command set  $S_i$ ,  $R_i$  denotes the number of read commands, while  $W_i$  denotes the number of write commands in it. Therefore, it is clear that we can write the relation between these notations as:

$$C_{RQ} = \sum_{i=1}^n C_{Si} = \sum_{i=1}^n (R_i + W_i) \quad (3)$$

Using these notations, the detail of the read-write aware throttling mechanism and the rank level read-write reordering are described in the following sections.



## 3.2 Read-Write Aware Throttling

The read-write aware throttling mechanism determines if a rank should be turned on whenever the throttle delay is reached. It checks on each command set  $S_i$ , which is composed of memory commands destined for  $r_i$  in the RQ, to see whether the condition  $R_i = 0$  is true. If the condition is satisfied,  $r_i$  is turned off and all the commands in  $S_i$  remain in the RQ for another throttle delay. On the other hand, if the condition is not satisfied,  $r_i$  is turned on and all the commands in  $S_i$  are allowed to be sent to CQ <sub>$i$</sub> .

The read-write aware throttling utilizes the fact that read requests affect system performance more than write requests [26]. It is performed on rank level and checks the existence of critical read requests in every command set whenever the throttle delay is reached. If a read request appears in a command set, the memory controller sets the target rank of this command set to *urgent*. Ranks with no pending read requests are set to *trivial*. All the commands destined for an urgent rank are allowed to be sent to the corresponding CQ, while other commands remain in the RQ for another throttle delay. This allows the memory controller to only turn on the urgent ranks and keep the trivial ranks in the low power mode, contributing to a large DRAM power saving.

To better understand how the read-write aware throttling mechanism works, we give a simple example. Suppose that there are four ranks in the DRAM. At a certain point, the throttle delay is reached and the memory commands are blocked inside the RQ, as shown in the left part of Fig. 9. Before sending the commands to the CQs, the commands are first clustered into command sets according to their target ranks. The set of commands destined for the same rank as the command enters the RQ first are reordered to the front. The set of commands destined for the same rank as the command that sits right after the command set at the front is reordered to second to the front, and so on. The order of the commands within the same command set remains the same, the command that enters the RQ earlier is closer to the front of the command

set. The request command sequence after clustering is shown in the right part of Fig. 9, and the pseudo code of clustering command sets is given below Fig. 9. In the pseudo code, the DRAM is assumed to have  $n$  ranks and  $cmd_j$  represents the command in the  $j^{\text{th}}$  slot in the RQ. The action *insert* in line 8 reorders a command to the  $cmdSetMark[i]^{\text{th}}$  slot and pushes all the commands behind it one slot towards the end of the RQ.

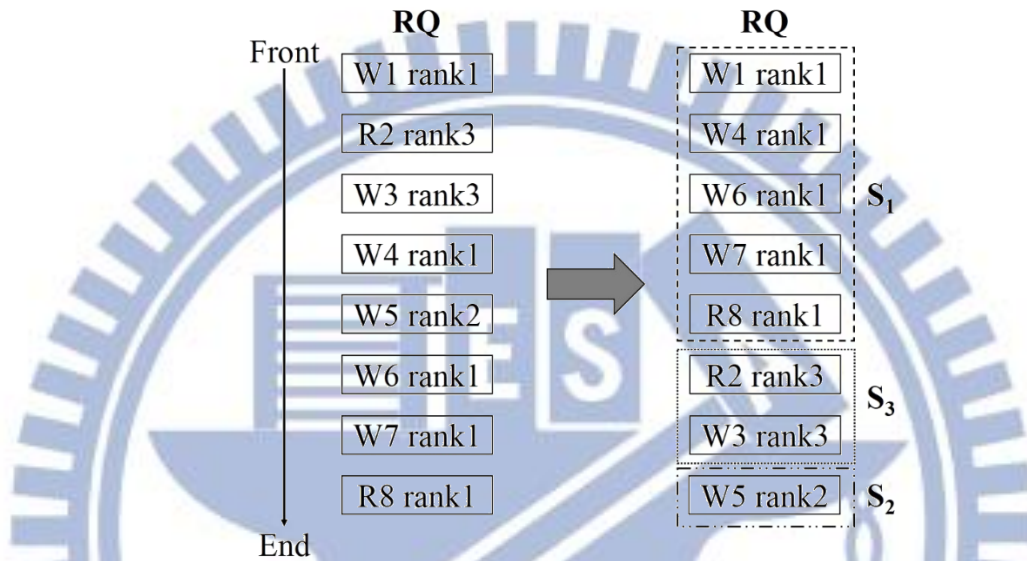


Fig. 9 An example of how commands blocked in the RQ are clustered into command sets.

```

function ClusterCommandSets( )
1. for  $l \leftarrow 1$  to  $n$  do
2.    $cmdSetMark[l] \leftarrow -1$ 
3. for  $j \leftarrow 1$  to  $C_{RQ}$  do
4.   // Let the target rank of  $cmd_j$  be  $i$ 
5.   if  $cmdSetMark[i] = -1$  then
6.      $cmdSetMark[i] \leftarrow j + 1$ 
7.   else
8.     insert  $cmd_j$  to  $cmdSetMark[i]$ 
9.   for  $k \leftarrow 1$  to  $n$  do
10.    if  $cmdSetMark[k] > cmdSetMark[i]$  then
11.       $cmdSetMark[k] \leftarrow cmdSetMark[k] + 1$ 
12.     $cmdSetMark[i] \leftarrow cmdSetMark[i] + 1$ 

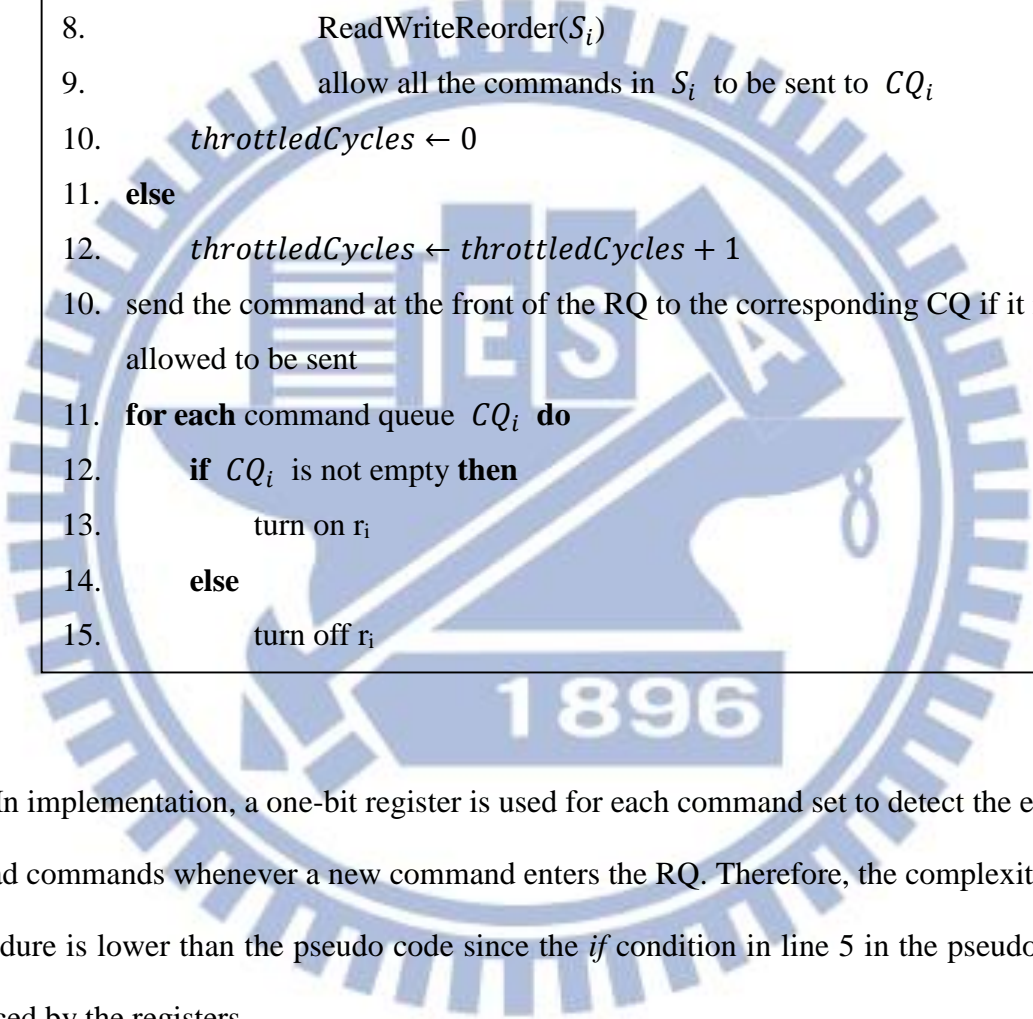
```

From Fig. 9, we can see that:

$$\begin{cases} C_{S_1} = 5, R_1 = 1, W_1 = 4 \\ C_{S_2} = 1, R_2 = 0, W_2 = 1 \\ C_{S_3} = 2, R_3 = 1, W_3 = 1 \\ C_{S_4} = 0, R_4 = 0, W_4 = 0 \end{cases}$$

The read-write aware throttling then checks for the command sets containing no read request commands. Since the command set  $S_4$  has no commands in it, its target rank  $r_4$  is considered to be trivial and is turned off to save power. The command set  $S_2$  contains no read requests and thus its target rank  $r_2$  is also considered trivial and is turned off. All the commands inside  $S_2$  are kept blocked in the RQ for another throttle delay. The ranks  $r_1$  and  $r_3$  are considered urgent because there are read requests in  $S_1$  and  $S_3$ . Therefore, commands in  $S_1$  and  $S_3$  are allowed to be sent to  $CQ_1$  and  $CQ_3$ , respectively. The memory controller turns on  $r_1$  and  $r_3$  to process the commands in  $S_1$  and  $S_3$ .

The pseudo code of the read-write aware throttling, is given as below. The function `ReadWriteReorder` in line 8 will be explained in the next section.



```

1. if throttledCycles =  $t_{TD}$  then
2.   ClusterCommandSets()
3.   for each command set  $S_i$  do
4.     urgentRank  $\leftarrow$  false
5.     if  $R_i \neq 0$  then
6.       urgentRank  $\leftarrow$  true
7.     if urgentRank is true then
8.       ReadWriteReorder( $S_i$ )
9.       allow all the commands in  $S_i$  to be sent to  $CQ_i$ 
10.    throttledCycles  $\leftarrow$  0
11.  else
12.    throttledCycles  $\leftarrow$  throttledCycles + 1
10. send the command at the front of the RQ to the corresponding CQ if it is
    allowed to be sent
11. for each command queue  $CQ_i$  do
12.   if  $CQ_i$  is not empty then
13.     turn on  $r_i$ 
14.   else
15.     turn off  $r_i$ 

```

In implementation, a one-bit register is used for each command set to detect the existence of read commands whenever a new command enters the RQ. Therefore, the complexity of this procedure is lower than the pseudo code since the *if* condition in line 5 in the pseudo code is replaced by the registers.



### 3.3 Rank Level Read-Write Reordering

The rank level read-write reordering is a scheduling policy for the command sets containing read requests in the RQ. It gives read requests higher priority than write requests. The commands in a command set are sent to the CQ in descending order of priority. Since the CQ issues commands to the DIMM in FIFO order, read requests are issued to the DIMM prior to write requests. This forces the DIMM inside the DRAM to process read requests, which are critical to system performance, as soon as possible. The rank level read-write reordering effectively relieves the system performance degradation caused by the DRAM power management policy.

The system performance degradation is greatly relieved if the read requests are sent to the CQ prior to all the write requests. However, reordering memory commands blindly can lead to a data hazard issue since the memory commands are no longer handled by the DIMM in the same order as the processors sent out. If a read request enters the RQ after a write request and they are both destined for a same address, a read-after-write (RAW) data hazard occurs if the DIMM returns data for this read request prior to the write request. To avoid RAW hazard, the rank level read-write reordering performs a check before reordering. For each read request in the a command set, the rank level read-write reordering checks all the write requests in the same command set that entered the RQ earlier than this read request. If one or more write requests target to the same address as the read request does, they are combined in their original order to form a *command group*. All the command groups are then reordered to preserve the FIFO order of the read request in each command group.

An example is given in Fig. 10, where each rectangle represents a memory command with its access type denoted by W (Write) or R (Read) followed by an index number and the target address of the command. The index numbers are assigned to each memory command according the time they entered the RQ. The command that enters the RQ earlier gets a smaller index

number. Fig. 10 illustrates how the commands in a command set  $S_1$  is combined into command groups and reordered.

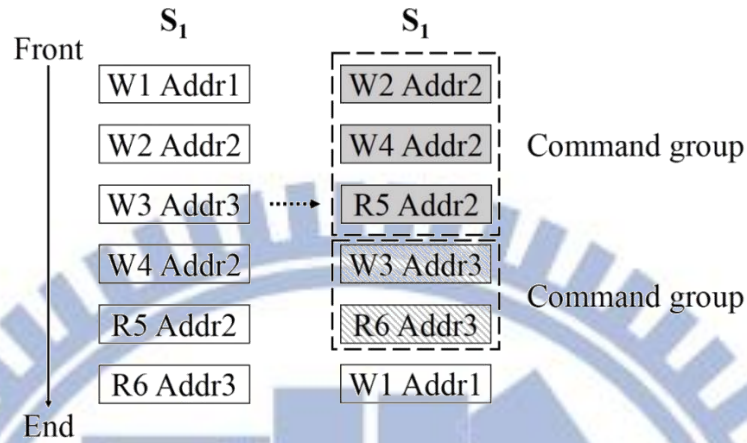


Fig. 10 An example of how commands in a given command set  $S_1$  are combined into command groups and then reordered.

As shown in Fig. 10, the rank level read-write reordering checks  $S_1$  for read requests. Command R5 is first found, and the rank level read-write reordering search through W1 to W4 to find that W2 and W4 have the same target address as R5. Therefore W2, W4, and R5 are combined into a command group, and the order of these three commands is preserved inside the command group. The rank level read-write reordering then finds R6 and W3, which have the same target address as R6. W3 and R6 are thus combined into a command group. Since R5 enters the  $S_1$  before R6, the command group containing R5 is placed in front of the command group containing R6. Commands that are not in any command group are placed in FIFO order behind the last command group.

After combining command groups and reordering, the reordered command sets are sent to the CQs. The target ranks of these command sets are turned on to process these commands. It is switched back to the low power mode once all the commands are finished and is kept in the low power mode until the throttle delay is once again reached.

The pseudo code of the rank level read-write reordering is given below. In the pseudo code, the command set  $S_i$  is assumed to have  $C_{S_i}$  commands and notations  $cmd_j$  and  $cmd_k$  represent the  $j^{\text{th}}$  and  $k^{\text{th}}$  command in the command set, respectively. Notice that in line 6 and line 9, the action *insert* reorders the command to the  $mark^{\text{th}}$  slot in the command set. For instance, line 6 moves  $cmd_k$  to the  $mark^{\text{th}}$  slot and all the commands originally sitting in the  $mark^{\text{th}}$  slot to the  $(k - 1)^{\text{th}}$  slot are shifted to  $(mark + 1)^{\text{th}}$  slot to  $k^{\text{th}}$  slot in order.

```

function ReadWriteReorder( $S_i$ ):
input: A command set  $S_i$ 
1.   $mark \leftarrow 1$ 
2.  for  $j \leftarrow 1$  to  $C_{S_i}$  do
3.      if  $cmd_j$  is a read request then
4.          for  $k \leftarrow mark$  to  $j - 1$  do
5.              if target address of  $cmd_k =$  target address of  $cmd_j$  then
6.                  insert  $cmd_k$  to  $mark$ 
7.                   $mark \leftarrow mark + 1$ 
8.              insert  $cmd_j$  to  $mark$ 
9.               $mark \leftarrow mark + 1$ 

```

As shown in the pseudo code, the rank level read-write reordering checks for read requests among all the commands in a command set from the front to the end of the command set. For each read request found, the rank level read-write reordering search through commands that are in front of the read request and do not belong to any command group. All the commands with the same target address as the read request are combined into a command group.

In implementation, the action *insert* is provided by the modern memory controller and no extra hardware is required. The address comparing action in line 5 uses a comparator, whose size differs from 1-bit to the length of the memory address. When a small comparator is used, the rank level read-write reordering is more conservative and tends to insert more write requests

in front of the read requests. The effect on relieving system performance degradation may be slightly weakened if a small comparator is used.





### 3.4 An Example of The Proposed Policy

In the proposed scheduling policy, the rank level read-write reordering is combined with the read-write aware throttling. To understand how the proposed policy works, a complete example is given as follows. Assume that there are four ranks in the DRAM and the throttle delay is reached at cycle  $k$ . Suppose that at cycle  $k$ , the command pattern in the RQ is as shown in the left part of Fig. 11. The commands are clustered into command sets according to their target ranks, the result is as shown in the right part of Fig. 11. Notice that the command set  $S_4$  contains no commands and is therefore omitted in Fig. 11.

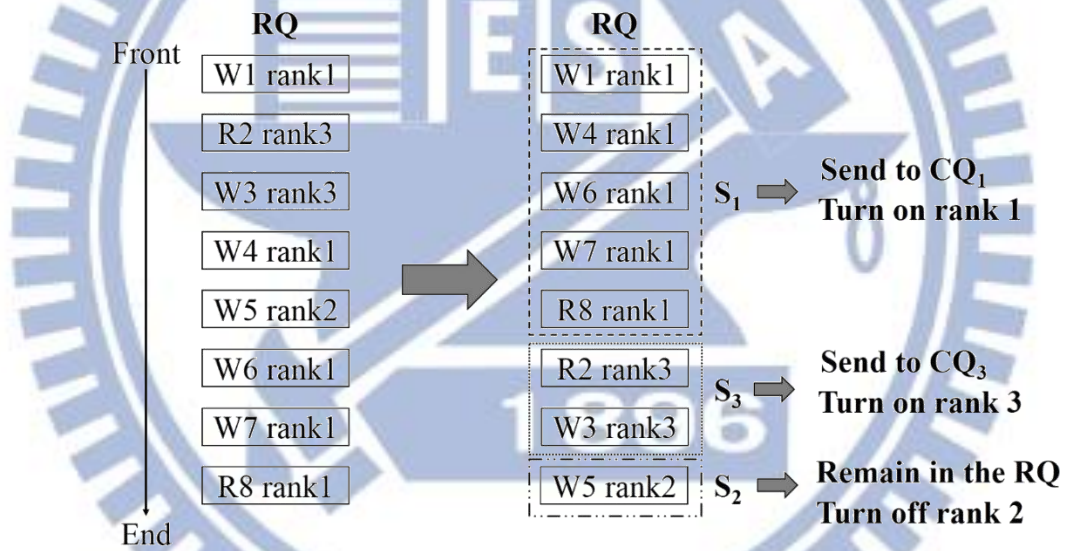


Fig. 11 An example of how the read-write aware throttling clusters commands in the RQ and determines which ranks should be turned on when the throttle delay is reached.

When the throttle delay is reached at cycle  $k$ , the read-write aware throttling is first performed to check on each command set for the existence of read commands. As the result,  $r_1$  and  $r_3$  are considered urgent, while  $r_2$  and  $r_4$  are considered trivial. After the read-write aware throttling, the rank level read-write reordering is performed on command sets  $S_1$  and  $S_3$  before they are sent to CQ<sub>1</sub> and CQ<sub>3</sub>. Using  $S_1$  as an example, Fig. 12 shows how the

commands in  $S_1$  are reordered and sent to the CQ. The command at the front of the command set has the highest priority while the command at the end has the lowest priority. The commands are sent to the CQ in descending order of priority one at a cycle.

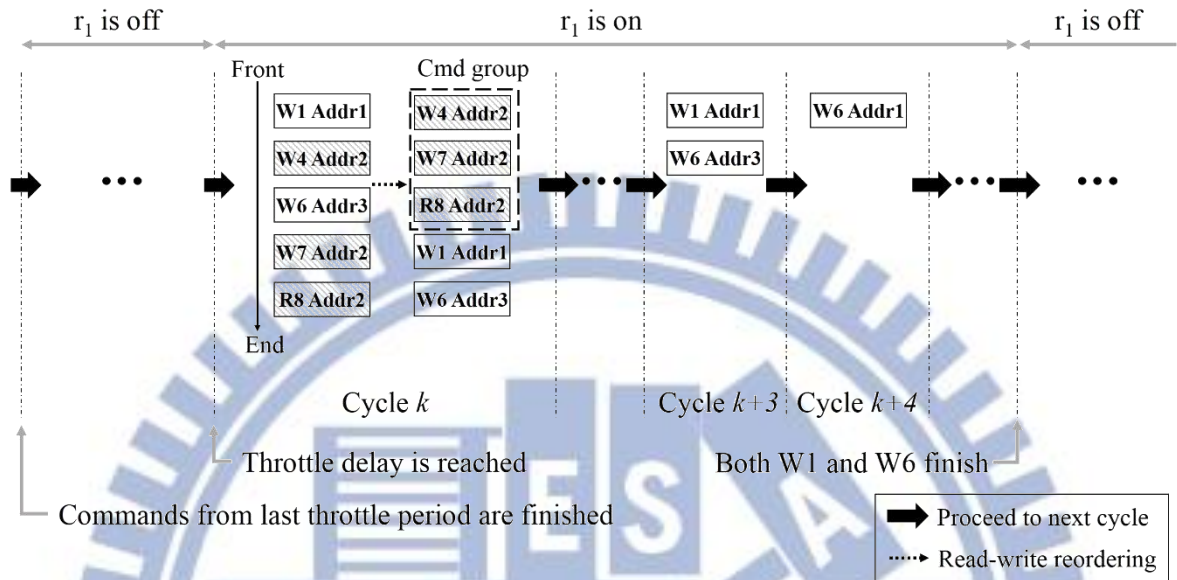


Fig. 12 An example of how the commands in a given command set  $S_1$  are reordered by the rank level read-write reordering and sent to the CQ.

As shown in Fig. 12, rank 1 was originally in the low power mode after all the commands from last throttle period are completed. When the throttle delay is reached and the command set is formed, the rank level read-write reordering checks through the command set and found that W4 and W7 target to the same address as R8. These three commands are then combined to form a command group and reordered to the front of  $S_1$ . Once the reordering completes, the commands are sent to  $CQ_1$  and rank 1 is turned on. Rank 1 is turned off again when all the command finishes. The example shows that the rank level read-write reordering forces the DRAM to process read requests as early as possible.

It is noticeable that for an urgent rank  $r_i$ , all the commands in the command set  $S_i$  are allowed to be sent to  $CQ_i$ . Even the write requests whose target addresses are not identical to any read requests are allowed to be sent to the corresponding CQ instead of kept blocked in the RQ. For example, the target addresses of W1 and W6 in Fig. 12 are not the same as R8, and

they are still sent to  $CQ_1$ . The reason is that once a rank is activated, keeping these write requests in the RQ does not contribute to more DRAM power reduction because they will eventually be processed and consumes the same amount of power. Moreover, if these write requests are kept blocked in the RQ until a read requests with the same target address enters the command set, the read request has to wait for these write requests to be completed. This lengthens the latency of the critical read request and has chance to worsen the system performance degradation.



# Chapter 4

## Experimental Results

This chapter demonstrates and analyzes the experimental results to examine the proposed techniques. First, the simulation environment is described. Second, the evaluation results of how each techniques proposed in this thesis work at certain throttle delay are shown and analyzed. Finally, the evaluation on the power and performance trade-off is carried out and the results are analyzed and compared with other works.

### 4.1 Simulation Environment

The performance of our work is evaluated with Multi2Sim [28], a widely used cycle-accurate system simulator. Multi2Sim provides detailed simulation of single core or multicore processors and gives us the statistics of the system performance. Our evaluation integrates DRAMSim2 [29] into Multi2Sim to obtain more accurate statistics of DRAM, such as the latency of each memory command, DRAM power consumption and power mode transition delays. DRAMSim2 is a cycle-accurate, JEDEC DDRx memory system simulator, which models the memory controller, memory channels, DRAM ranks, and banks. In an evaluation, Multi2Sim runs the benchmark and generates memory commands accordingly. The memory commands are sent to DRAMSim2 and processed by the DRAM that DRAMSim2 models. The evaluation results of Multi2Sim provide the system throughput statistics, while the evaluation results of DRAMSim2 provide the DRAM power consumption.

The baseline system in the simulations uses the ARM Coretex-A9 MPCore [30]. The configuration parameters of ARM Coretex-A9 MPCore are listed in Table III. The baseline system has two-level caches. In order to evaluate our work by comparing to the previous work



[21], our simulations use the same cache sizes as in the previous work. The detail parameters of the memory system are presented in Table IV. The main memory used in the evaluation is a DDR2 SDRAM, which is one of the JEDEC standard memory available on market [12].

Table III  
Configuration parameters of ARM Cortex A9 [30]

| <i>Parameter</i>                          | <i>Value</i>                |
|---|-----------------------------|
| Number of cores                           | 4                           |
| Number of threads per core                | 1                           |
| Technology node                           | 40 nm                       |
| Operating frequency                       | 2.132 GHz                   |
| Supply voltage                            | 0.66 V                      |
| Threshold voltage                         | 0.23 V                      |
| Decode width                              | 2                           |
| Issue width                               | 4                           |
| Commit width                              | 4                           |
| Number of arithmetic logic units per core | 3                           |
| Number of multipliers per core            | 1                           |
| Number of floating-point units per core   | 1                           |
| Branch predictor                          | 2 level, 1024-set 2-way BTB |

Table IV  
Memory system parameters

| <i>Parameter</i>  | <i>Value</i>   |
|---|----------------|
| Size of level 1 data cache per core                     | 32 KB          |
| Set associativity of level 1 data cache per core        | 4-way          |
| Size of level 1 instruction cache                       | 64 KB          |
| Set associativity of level 1 instruction cache per core | 2-way          |
| Size of level 2 cache                                   | 2 MB           |
| Set associativity of level 2 cache                      | 8-way          |
| <b>DRAM frequency</b>                                   | <b>533 MHz</b> |
| <b>Number of DRAM ports</b>                             | <b>2</b>       |
| <b>DRAM device width</b>                                | <b>8</b>       |
| <b>Number of DRAM ranks</b>                             | <b>4</b>       |
| <b>Number of DRAM banks per rank</b>                    | <b>4</b>       |
| <b>Number of DRAM rows per bank</b>                     | <b>8192</b>    |
| <b>Number of DRAM columns per bank</b>                  | <b>4096</b>    |

Table V  
Benchmark combinations of floating-point benchmarks in SPEC CPU2006 [31]

| <i>Combination</i> | <i>Benchmarks</i> |               |              |              |
|--------------------|-------------------|---------------|--------------|--------------|
| fp1                | 410.bwaves        | 416.gamess    | 433.milc     | 434.zeusmp   |
| fp2                | 435.gromacs       | 436.cactusADM | 437.leslie3d | 444.namd     |
| fp3                | 447.dealII        | 450.soplex    | 453.povray   | 454.calculix |
| fp4                | 459.GemsFDTD      | 465.tonto     | 470.lbm      | 481.wrf      |

The workload for our simulations is the SPEC CPU2006 [31] benchmark suite. The benchmarks in the SPEC CPU2006 suite can be separated into integer benchmarks and floating point benchmarks. The floating point benchmarks have higher memory pressure than integer benchmarks and need more sophisticated power management policies [1][21]. Therefore, the benchmarks used in our simulations are randomly chosen from the floating point benchmarks.

For each simulation, a benchmark combination containing four benchmarks is used. Every benchmark in the benchmark combination is assigned to a certain core. The benchmark combinations are listed in Table V. Each benchmark combination is run for five million CPU cycles in our evaluation.

Another workload used in the evaluation is the SPLASH-2 benchmarks [32], which are collected from real applications. Using the dynamic context scheduler provided by Multi2Sim [28], each program in the SPLASH-2 benchmarks forks at most four parallel contexts during runtime. The benchmarks in SPLASH-2 used in the evaluation are listed in Table VI.

Table VI  
SPLASH-2 [32] benchmarks used in the evaluation

| <i>Benchmark</i> | <i>Problem size</i>                     |
|------------------|---|
| Barnes           | 2048 particles                          |
| Cholesky         | tk14.O                                  |
| FFT              | 65536 points                            |
| FMM              | 2048 particles                          |
| Radix            | 256k keys, max-value 524288, radix 4096 |

The SPEC CPU2006 benchmarks are used in section 4.2 and 4.3, and the SPLASH-2 benchmarks are used in section 4.3. In the evaluation, the power consumption is measured in Watts. The system performance is measured in million instructions per second (MIPS), which represents the throughput of the system. All the results are normalized to the *native DRAM*, which refers to the DRAM with no power management policy.

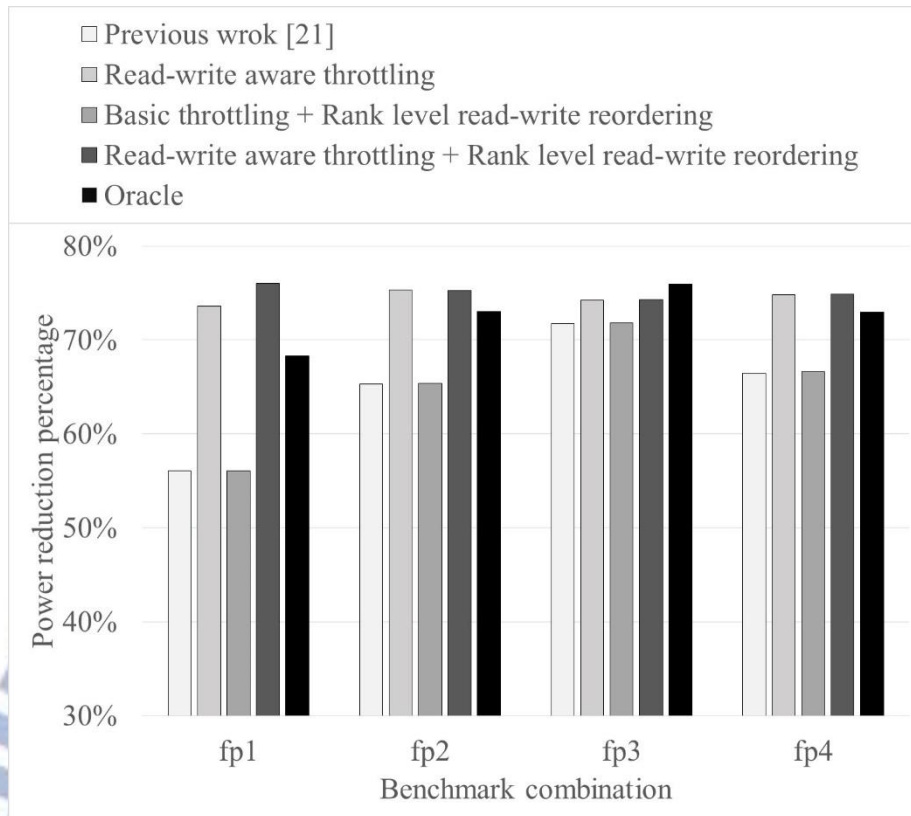
## 4.2 Analysis on Different Techniques

Although our work employs both the read-write aware throttling and the rank level read-write reordering, these two techniques can be employed individually. Therefore, the techniques proposed in this thesis are not only evaluated jointly but also separately to see how they affect the DRAM power consumption and the system performance.

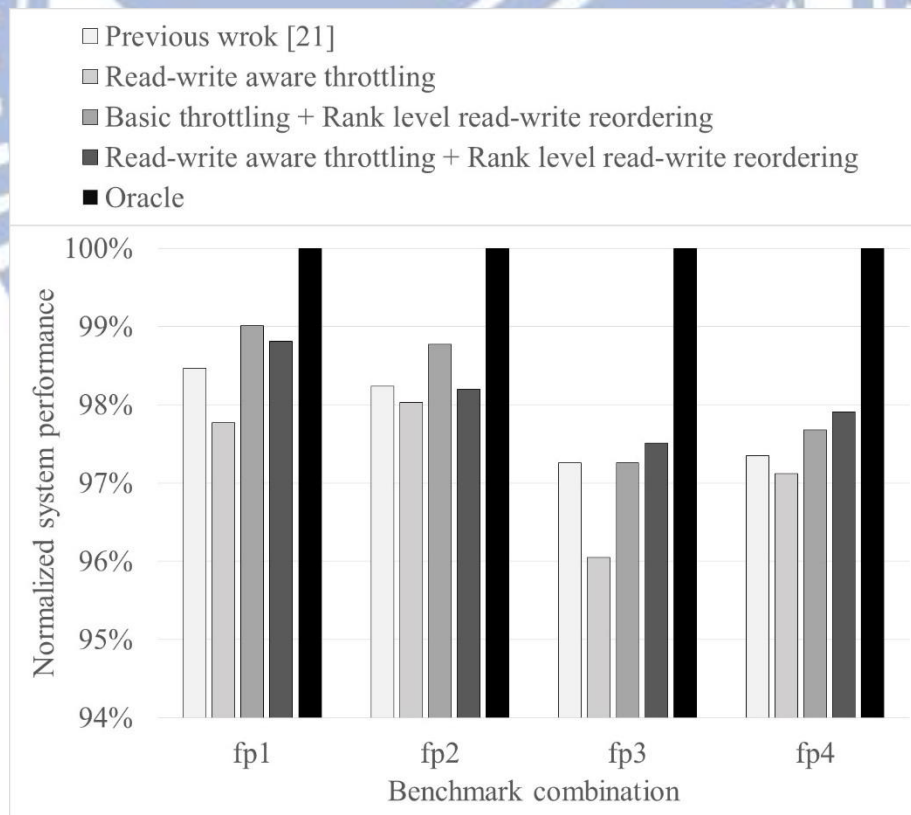
In the evaluation, our work is compared to the previous work and an oracle policy. In the oracle policy, the order of the memory accesses is transparent so that the DRAM ranks can be ideally turned on and off when needed. Furthermore, there is no transition delay and transition power in the oracle policy. The power reduction of the oracle policy is the maximum power reduction possible at zero system performance degradation. The oracle policy does not employ any throttling-based mechanism nor reordering. Therefore, the oracle policy can be viewed as a time-out-zero policy, which turns off a rank at the instant it becomes idle, with perfect pre-wakeup capability that turns a rank back on whenever it receives a command.

The benchmark combinations listed in Table V are used in this evaluation. The throttle delay for our work and the previous work is set to 400 CPU cycles, at which both our work and the previous work achieve good power reduction with acceptable system performance degradation. The effect of different throttle delays is analyzed later. The evaluation results of all four benchmark combinations are shown in Fig. 13.





(a) DRAM power reduction percentage



(b) Normalized system performance

Fig. 13 Power and performance of different policies on different benchmark combinations.

The evaluation results show that when the read-write aware throttling is employed alone, it reduces the DRAM power consumption 10%~15% more than the previous work but causes around 1% more system performance degradation. The reason is that the read-write aware throttling puts a rank into the low power mode until it receives read requests. However, when the rank is turned back on to handle the read requests, there are many write requests waiting to be processed. Without the rank level read-write reordering, the read requests have to wait until all the write requests that enters the queue before them are completed. The system performance is degraded since the critical read requests have to wait for a long time.

On the other hand, when the read-write reordering is used alone with the basic throttling mechanism as in the previous work [21], it improves the system performance by around 1% but the DRAM power consumption remains the same as the previous work [21]. This is because that the rank level read-write reordering only forces the DRAM to process read requests as early as possible and does not create extra power down opportunity.

By combining these two techniques, our work saves 10%~15% more power than the previous work with the same, or even slighter, system performance degradation. More importantly, our work reduces DRAM power consumption to below the oracle solution with 2% of the system performance degradation on average.

The evaluation results show that each technique reacts differently to different benchmark combinations. Since the proposed techniques take into consideration that read requests and write requests are not equally critical to the system performance, the number of requests contained in a benchmark is essential to the effect of the proposed techniques. Therefore, the read requests percentage of each benchmark combination are listed in Table VII. The read requests percentage is obtained by evaluating each benchmark combination with the native DRAM and is calculated by dividing the number of read requests into the total number of memory requests commands.

Table VII

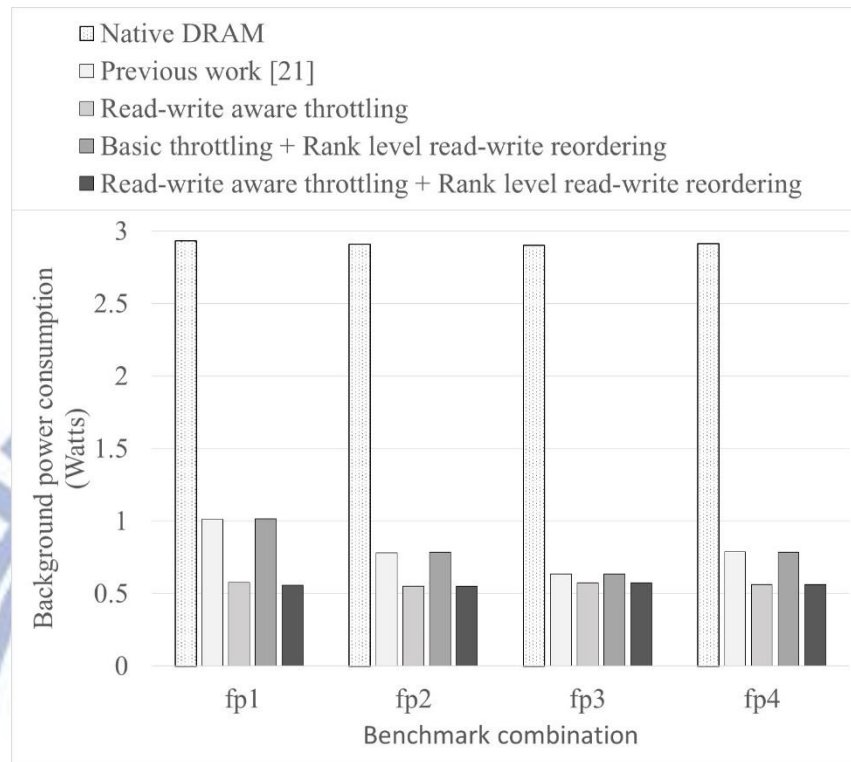
Read requests percentage of each benchmark combination

| <i>Benchmark combination</i> | <i>Read requests percentage</i> |
|------------------------------|---------------------------------|
| fp1                          | 6.76%                           |
| fp2                          | 17.74%                          |
| fp3                          | 55.27%                          |
| fp4                          | 20.14%                          |

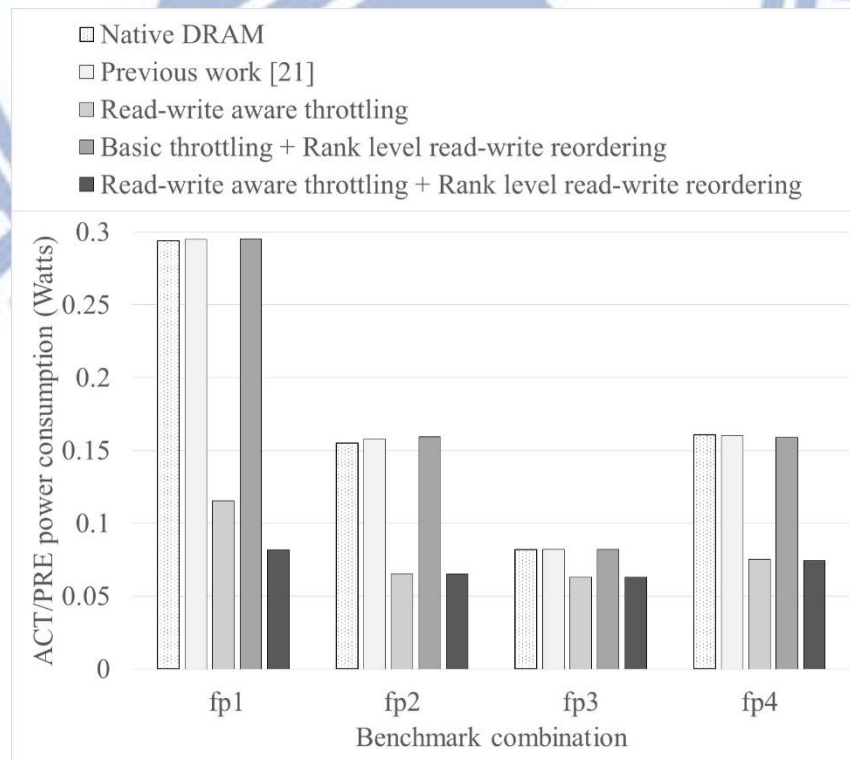
Table VII shows that most of the read requests are completed in the cache, and the read requests that send down to the DRAM is less than write requests. It is obvious that fp3 is the most read intensive benchmark combination, while the fp1 is the least read intensive one. The read intensity of different benchmark combination reflects on the power reduction in the evaluation results. The read-write aware throttling works very well on fp1, which has a weak read intensity, due to the fact that most of the time there are only write requests blocked in the RQ and DRAM ranks can be turned off. On the other hand, the strong read intensity of fp3 limits the effect of the read-write aware throttling since it is less likely for a rank to only receive write request in a throttle period and thus cannot be turned off. Nevertheless, our work still manages to save 5% more DRAM power consumption with slightly better system performance than the previous work [21].

As mentioned in section 1.2, the DRAM power can be partitioned into several parts, including background power, active power, precharge power, read power, write power, and the refresh power. To further analyze the evaluation results, Fig. 14 shows these detail power consumptions obtained from the evaluation and compare them to the power consumption of the native DRAM. In Fig. 14, the ACT/PRE power consumption represents the sum of active power and precharge power, while the Read/write burst power represents the summation of read power and write power. Notice that the refresh power is omitted because the DRAM is refreshed

periodically and the evaluation runs the benchmark for a fixed CPU cycle period, the refresh power consumptions for different techniques are the same.

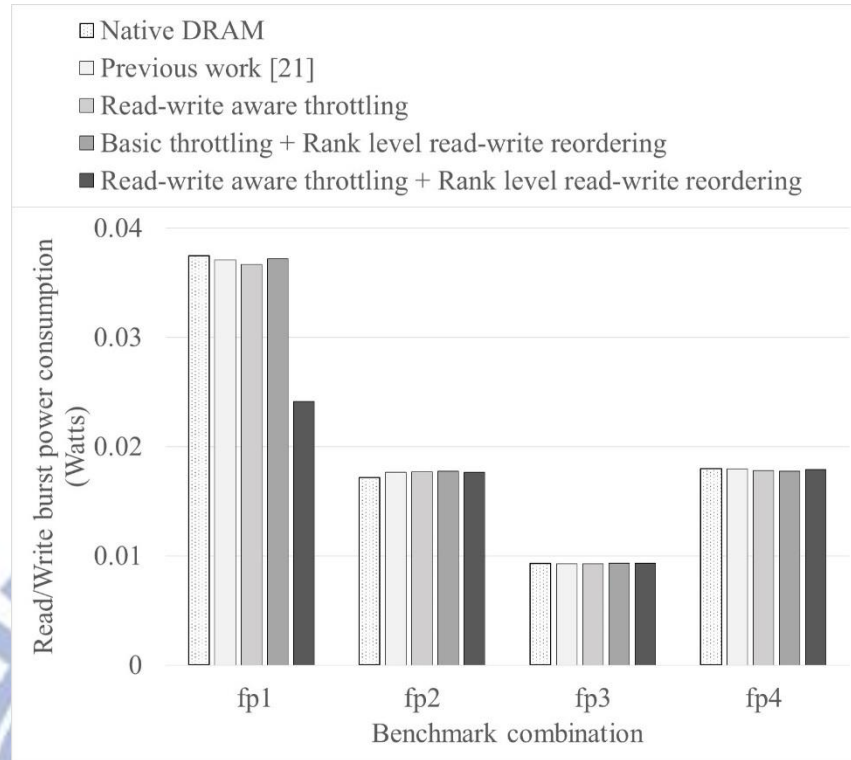


(a) Background power consumptions of different techniques



(b) ACT/PRE power consumptions of different techniques





(c) Read/write burst power consumptions of different techniques

Fig. 14 The background power, ACT/PRE power, and the read/write power consumptions of different techniques.

The results in Fig. 14 show that the DRAM power consumption is dominated by the background power since the main memory accesses are sparse in the evaluation. Therefore, the throttling-based mechanism is used to turn off idle ranks, and the background power consumption of DRAM is greatly reduced. In addition, the read-write aware throttling creates longer idle period for a rank to be turned off and thus reduces 10%~45% more of the background power against the previous work [21]. The ACT/PRE power consumptions and the read/write burst power consumptions show that the read-write aware throttling cuts down the number of returned commands because the DRAM ranks are in the low power mode for a long period. However, when the read-write aware throttling releases the blocked commands from the RQ to the CQ, it forces the DRAM to focus on accessing the active rank.

It is noticeable that the read/write burst power consumption of our work on fp1 is low. It is because that fp1 has a weak read intensity. When the memory controller finally receives a

read request and turns on a rank, there are many pending write requests targeting that rank, which are blocked by the read-write aware throttling. As the result, in a command group formed by rank level read-write reordering, there are many write requests in front of the read request. Once the rank is activated, it takes a long time processing the pending write requests before it becomes available to process the critical read request. The system performance is thus harmed and fewer main memory commands are completed within the same simulation period. Therefore, the read/write burst power consumption is lower than other techniques.



### 4.3 Power and Performance Trade-Off

In the proposed techniques, the read-write aware throttling mechanism is the main contributor to the DRAM power reduction. For throttling based power reduction mechanisms, the *throttle delay* is critical to both the DRAM power consumption and the system performance. Long throttle delay leads to better power reduction since the DRAM ranks stay in the low power mode for a long period. However, long throttle delay also leads to worse impact on the system performance because all the commands have to wait for a long period before they are processed by the DRAM. On the other hand, short throttle delay allows the DRAM to process memory commands more frequently but it also limits the effectiveness on the power reduction.

In order to see how different throttle delays affect the performance of our work, an evaluation on different throttle delays is carried out. The evaluation uses all benchmark combinations fp1, fp2, fp3 and fp4. Since every benchmark combination has a different memory command pattern, we averaged the evaluation results of all four benchmark combinations. The average simulation results are shown in Table VIII. All the throttle delays are in CPU cycles. The improvements are the differences between our work and the previous work [21].

Table VIII  
Effect of different throttle delays

| Throttle Delay | Power reduction percentage |          |             | System performance overhead |          |             |
|----------------|----------------------------|----------|-------------|-----------------------------|----------|-------------|
|                | Previous work [21]         | Our work | Improvement | Previous work [21]          | Our work | Improvement |
| 100            | 64.67%                     | 75.46%   | 10.79%      | 1.03%                       | 0.61%    | 0.42%       |
| 200            | 64.54%                     | 75.26%   | 10.72%      | 0.87%                       | 0.72%    | 0.15%       |
| 400            | 64.58%                     | 75.10%   | 10.52%      | 1.98%                       | 1.73%    | 0.25%       |
| 800            | 65.32%                     | 75.14%   | 9.82%       | 5.03%                       | 4.88%    | 0.15%       |
| 1600           | 65.49%                     | 74.96%   | 9.47%       | 6.46%                       | 6.39%    | 0.07%       |
| 3200           | 68.24%                     | 75.17%   | 6.93%       | 7.63%                       | 7.46%    | 0.17%       |
| 6400           | 71.02%                     | 75.42%   | 4.40%       | 7.92%                       | 7.88%    | 0.04%       |

The evaluation results show that our work is stable with different throttle delays. It steadily reduces around 75% of DRAM power consumption, which is an upper bound of power reduction, and is around 10% better than the previous work. Moreover, the system performance degradation of our work is slightly better than the previous work. This shows that the read-write aware reordering mechanism effectively relieves the impact on the system performance. The evaluation results also shows as the throttle delay increases, the difference in power reduction between our work and the previous work gets smaller. It is because that when the throttle delay is too large, all the DRAM ranks are in the low power mode for most of the time. Therefore the power consumption is low and the performance degradation is dramatic.

With the results in Table VIII, we can further illustrates the trade-off characteristic between power and performance. By varying the throttle delay, the evaluation shows how our work reacts to different system performance degradations. The average results of all benchmark combinations are shown in Fig. 15, where both the power and performance are normalized to the DRAM with no power management policy. The result shows that our work has a better



power and performance trade-off characteristic. Under the same system performance degradation, our work reduces around 10% more of DRAM power than the previous work.

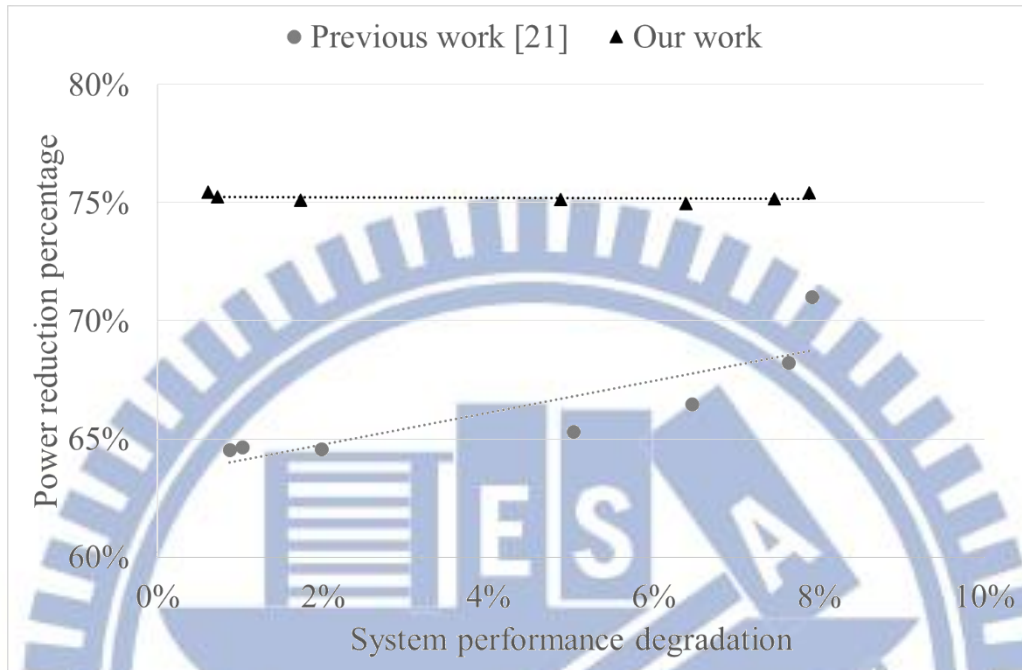


Fig. 15 Average power and performance trade-off characteristics on SPEC CPU2006 [31].

It is noticeable that our work, unlike the previous work [21], is sensitive to the memory command patterns of benchmarks. To show the difference, the evaluation results of benchmark combinations fp1 and fp3 are shown in Fig. 16 and Fig. 17 respectively. The number of read requests is much larger than write requests in fp3, while write requests dominates over read requests in fp1.

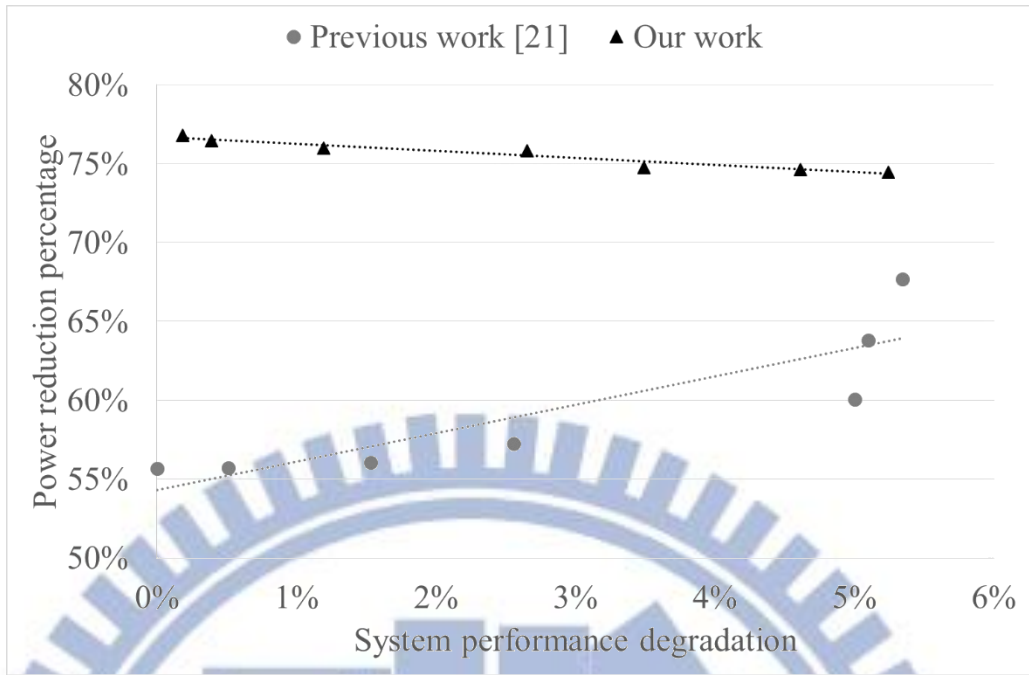


Fig. 16 Power and performance trade-off characteristics for fp1.

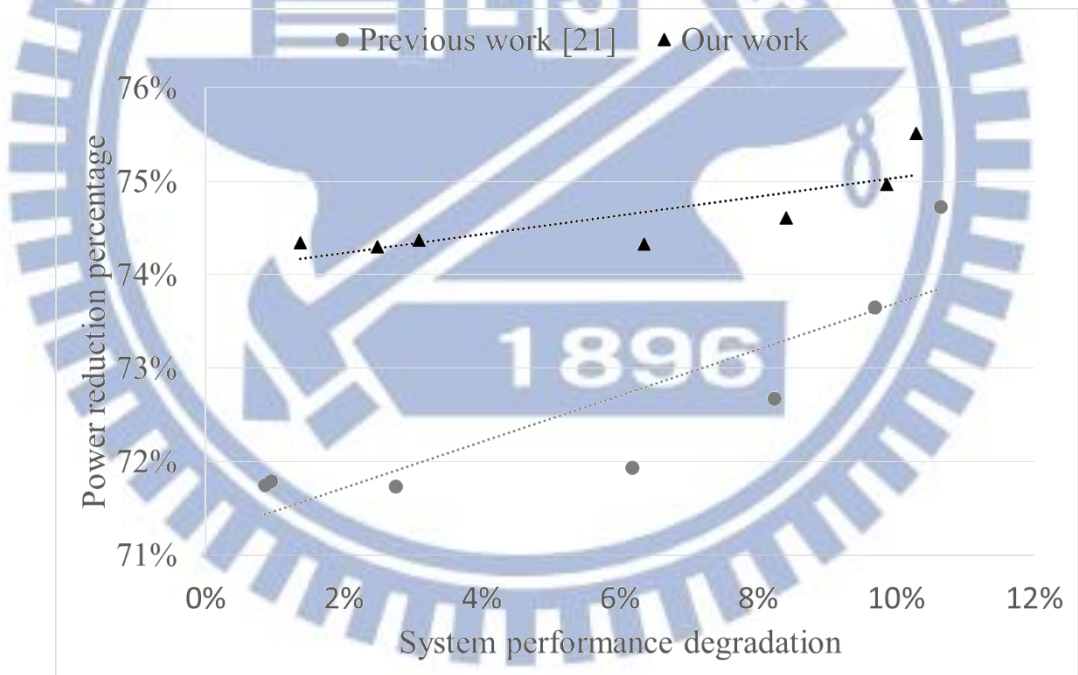


Fig. 17 Power and performance trade-off characteristics for fp3.

The trade-off curves of the previous work [21] in Fig. 16 and Fig. 17 have similar slopes. On the other hand, the slope of trade-off curves of our work are different for fp1 and fp3. The results show that when the application is more read intensive, our work is able to effectively reduce more power when with a slight more system performance degradation. The reason is

that, the read-write aware throttling accumulates write accesses in the RQ until a read access appears. For read intensive applications, only a few write accesses are kept in the RQ by the read-write aware throttling. Therefore, a slight increment in system performance degradation indicates that the throttle delay is greatly lengthened, which also leads to a much better power reduction.

In order to evaluate how our work performs when the context dynamically forks out, the evaluation on the SPLASH-2 benchmarks [32] is carried out and the resulting trade-off curves are shown in Fig. 18. Both the power reduction and the system performance degradation are normalized to the native DRAM. The system performance here is measured by the number of cycles used to complete a benchmark.

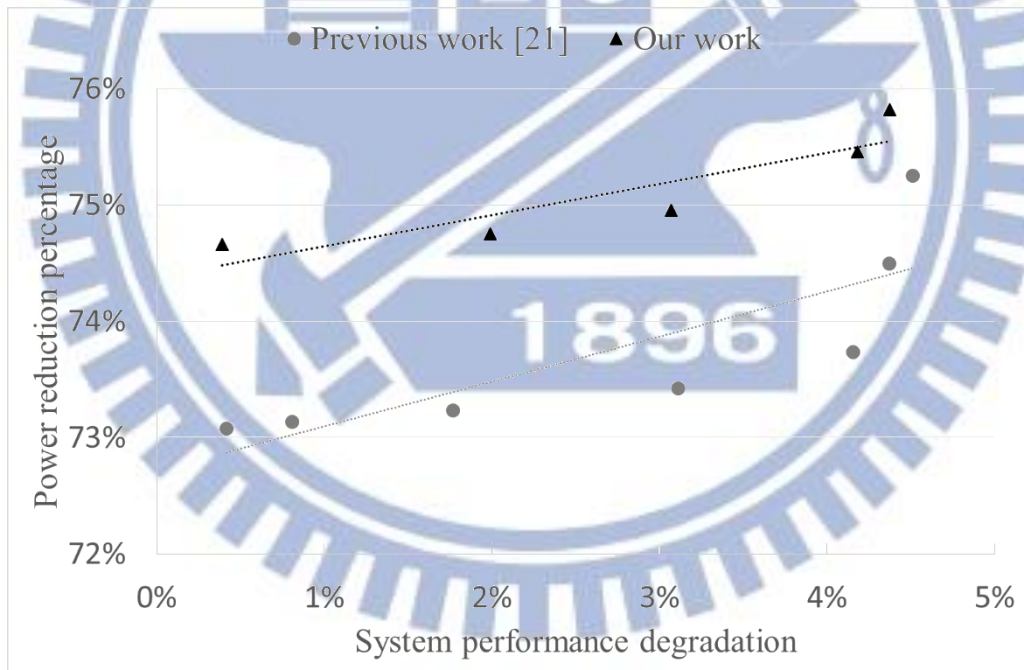


Fig. 18 Average power and performance trade-off characteristics on SPLASH-2 [32].

The trade-off curves in Fig. 18 show that the difference between our work and the previous work [21] is smaller than the results obtained from running SPEC CPU2006 benchmarks. The reason is that the SPLASH-2 benchmarks are not memory intensive comparing to the SPEC CPU2006 benchmarks. As the result, the DRAM is turned off most of the time during evaluation

for both the previous work [21] and our work. However, our work still achieves higher power reduction under the same system performance. The statistics of main memory requests per million cycles of different benchmark combinations and benchmarks are listed in Table IX to show the memory intenseness.

Table IX  
Main memory requests per million cycles of different benchmarks

|                   | <i>Benchmark combinations/<br/>benchmarks</i> | <i>Main memory requests<br/>per million cycles</i> |
|-------------------|---|--|
| SPEC CPU2006 [31] | fp1   | 511391.80  |
|                   | fp2   | 818090.00  |
|                   | fp3   | 31697.60   |
|                   | fp4   | 442664.80  |
| SPLASH-2 [32]     | cholesky                                      | 894.36   |
|                   | fft   | 2196.41  |
|                   | fmm   | 271.93   |
|                   | radix   | 801.45   |
|                   | barnes  | 38.60  |

The detail evaluation results of each benchmark combination are shown in Table X. The improvement section shown in Table X is normalized to the previous work. The detail evaluation results show that for a variety of applications, our work provides a superior power reduction at the cost of a minor system performance degradation.



Table X

Detail evaluation results on different throttle delays for SPEC CPU2006 [31]

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | MIPS      | Power (Watts) | MIPS      | Power       | MIPS   |
| 100            | 1.5194             | 1751.8644 | 0.7952        | 1748.6664 | 47.67%      | -0.18% |
| 200            | 1.5186             | 1742.9100 | 0.8070        | 1745.0420 | 46.86%      | 0.12%  |
| 400            | 1.5060             | 1725.0012 | 0.8230        | 1730.9708 | 45.36%      | 0.35%  |
| 800            | 1.4654             | 1707.0924 | 0.8282        | 1705.3868 | 43.48%      | -0.10% |
| 1600           | 1.3687             | 1664.2392 | 0.8648        | 1690.6760 | 36.81%      | 1.59%  |
| 3200           | 1.2401             | 1662.5336 | 0.8692        | 1671.0616 | 29.91%      | 0.51%  |
| 6400           | 1.1081             | 1658.2696 | 0.8749        | 1659.9752 | 21.04%      | 0.10%  |

(a) fp1

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | MIPS      | Power (Watts) | MIPS      | Power       | MIPS   |
| 100            | 1.1158             | 2094.9032 | 0.7973        | 2134.1320 | 28.54%      | 1.87%  |
| 200            | 1.1250             | 2125.1776 | 0.8050        | 2131.3604 | 28.44%      | 0.29%  |
| 400            | 1.1288             | 2107.0556 | 0.8048        | 2106.2028 | 28.71%      | -0.04% |
| 800            | 1.0938             | 2010.9024 | 0.7961        | 2049.7048 | 27.21%      | 1.93%  |
| 1600           | 1.0828             | 2025.6132 | 0.7952        | 2029.0244 | 26.56%      | 0.17%  |
| 3200           | 1.0446             | 1998.9632 | 0.7856        | 2010.4760 | 24.80%      | 0.58%  |
| 6400           | 0.9436             | 1995.3388 | 0.7775        | 1988.9428 | 17.60%      | -0.32% |

(b) fp2

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | MIPS      | Power (Watts) | MIPS      | Power       | MIPS   |
| 100            | 0.8920             | 2464.5920 | 0.8091        | 2409.1600 | 9.30%       | -2.25% |
| 200            | 0.8905             | 2462.4600 | 0.8101        | 2451.8000 | 9.03%       | -0.43% |
| 400            | 0.8922             | 2417.6880 | 0.8113        | 2424.0840 | 9.07%       | 0.26%  |
| 800            | 0.8861             | 2332.4080 | 0.8105        | 2328.1440 | 8.53%       | -0.18% |
| 1600           | 0.8626             | 2281.2400 | 0.8018        | 2276.9760 | 7.05%       | -0.19% |
| 3200           | 0.8318             | 2244.9960 | 0.7903        | 2240.7320 | 4.98%       | -0.19% |
| 6400           | 0.7977             | 2221.5440 | 0.7732        | 2230.0720 | 3.08%       | 0.38%  |

(c) fp3

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | MIPS      | Power (Watts) | MIPS      | Power       | MIPS   |
| 100            | 1.1234             | 2067.4004 | 0.8088        | 2048.0300 | 28.01%      | -0.94% |
| 200            | 1.1332             | 2076.3548 | 0.8153        | 2083.6036 | 28.05%      | 0.35%  |
| 400            | 1.1350             | 2041.8164 | 0.8194        | 2035.5424 | 27.80%      | -0.31% |
| 800            | 1.1177             | 1989.7956 | 0.8204        | 1980.8412 | 26.59%      | -0.45% |
| 1600           | 1.0923             | 1948.2216 | 0.8177        | 1940.5464 | 25.14%      | -0.39% |
| 3200           | 1.0551             | 1917.3076 | 0.8079        | 1910.2720 | 23.42%      | -0.37% |
| 6400           | 0.9556             | 1914.5360 | 0.7955        | 1914.5360 | 16.75%      | 0.00%  |

(d) fp4

Table XI  
Detail evaluation results on different throttle delays for SPLASH-2 [32]

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | Cycles    | Power (Watts) | Cycles    | Power       | Cycles |
| 100            | 0.86591            | 386216401 | 0.80595       | 383973995 | 6.92%       | 0.58%  |
| 200            | 0.86641            | 386747351 | 0.80678       | 382694644 | 6.88%       | 1.05%  |
| 400            | 0.86339            | 390881890 | 0.80490       | 384898787 | 6.77%       | 1.53%  |
| 800            | 0.85677            | 394132765 | 0.80304       | 390673931 | 6.27%       | 0.88%  |
| 1600           | 0.84823            | 398198400 | 0.79715       | 396053325 | 6.02%       | 0.54%  |
| 3200           | 0.82893            | 398909581 | 0.78876       | 398746298 | 4.85%       | 0.04%  |
| 6400           | 0.80316            | 400814358 | 0.77917       | 399440076 | 2.99%       | 0.34%  |

(a) cholesky

| Throttle delay | Previous work [21] |            | Our work      |            | Improvement |        |
|----------------|--------------------|------------|---------------|------------|-------------|--------|
|                | Power (Watts)      | Cycles     | Power (Watts) | Cycles     | Power       | Cycles |
| 100            | 1.18733            | 1626732828 | 1.06719       | 1599358623 | 10.12%      | 1.68%  |
| 200            | 1.18243            | 1646152322 | 1.06244       | 1613998522 | 10.15%      | 1.95%  |
| 400            | 1.17409            | 1680569237 | 1.06071       | 1630723965 | 9.66%       | 2.97%  |
| 800            | 1.15385            | 1728206075 | 1.05251       | 1677411431 | 8.78%       | 2.94%  |
| 1600           | 1.11851            | 1783437872 | 1.03031       | 1740400454 | 7.89%       | 2.41%  |
| 3200           | 1.03481            | 1785862617 | 0.97675       | 1769778925 | 5.61%       | 0.90%  |
| 6400           | 0.97555            | 1793418974 | 0.94211       | 1781048450 | 3.43%       | 0.69%  |

(b) fft

| Throttle delay | Previous work [21] |            | Our work      |            | Improvement |        |
|----------------|--------------------|------------|---------------|------------|-------------|--------|
|                | Power (Watts)      | Cycles     | Power (Watts) | Cycles     | Power       | Cycles |
| 100            | 0.70985            | 1140576902 | 0.69463       | 1138116645 | 2.14%       | 0.22%  |
| 200            | 0.70761            | 1144003008 | 0.69385       | 1139193870 | 1.94%       | 0.42%  |
| 400            | 0.70981            | 1149568000 | 0.69341       | 1141876406 | 2.31%       | 0.67%  |
| 800            | 0.70855            | 1151543397 | 0.69405       | 1150034077 | 2.05%       | 0.13%  |
| 1600           | 0.70548            | 1159231177 | 0.69312       | 1154615545 | 1.75%       | 0.40%  |
| 3200           | 0.70202            | 1158699823 | 0.68899       | 1160365744 | 1.86%       | -0.14% |
| 6400           | 0.6924             | 1161981653 | 0.68676       | 1159341940 | 0.81%       | 0.23%  |

(c) fmm

| Throttle delay | Previous work [21] |           | Our work      |           | Improvement |        |
|----------------|--------------------|-----------|---------------|-----------|-------------|--------|
|                | Power (Watts)      | Cycles    | Power (Watts) | Cycles    | Power       | Cycles |
| 100            | 0.83416            | 567929757 | 0.79279       | 561936504 | 4.96%       | 1.06%  |
| 200            | 0.83118            | 569758770 | 0.79116       | 563959080 | 4.81%       | 1.02%  |
| 400            | 0.82504            | 575787691 | 0.78821       | 568383335 | 4.46%       | 1.29%  |
| 800            | 0.82154            | 591603316 | 0.78341       | 584040811 | 4.64%       | 1.28%  |
| 1600           | 0.81879            | 592267385 | 0.77909       | 581449146 | 4.85%       | 1.83%  |
| 3200           | 0.80244            | 596695273 | 0.76463       | 595039745 | 4.71%       | 0.28%  |
| 6400           | 0.77625            | 592663748 | 0.75247       | 597326751 | 3.06%       | -0.79% |

(d) radix

| Throttle delay | Previous work [21] |            | Our work      |            | Improvement |        |
|----------------|--------------------|------------|---------------|------------|-------------|--------|
|                | Power (Watts)      | Cycles     | Power (Watts) | Cycles     | Power       | Cycles |
| 100            | 0.65358            | 1146352684 | 0.65019       | 1146958803 | 0.52%       | -0.05% |
| 200            | 0.65372            | 1146377562 | 0.65036       | 1145273539 | 0.51%       | 0.10%  |
| 400            | 0.65382            | 1146687597 | 0.6501        | 1143846545 | 0.57%       | 0.25%  |
| 800            | 0.65412            | 1146328710 | 0.64973       | 1145234276 | 0.67%       | 0.10%  |
| 1600           | 0.65343            | 1145896274 | 0.64998       | 1147168394 | 0.53%       | -0.11% |
| 3200           | 0.65286            | 1146058867 | 0.64953       | 1148778795 | 0.51%       | -0.24% |
| 6400           | 0.65252            | 1147747030 | 0.64993       | 1146213644 | 0.40%       | 0.13%  |

(e) barnes



# Chapter 5

## Conclusions and Future Works

This thesis proposes a DRAM scheduling policy to magnificently reduce the power consumption of DRAM. The read-write aware throttling mechanism allows the DRAM ranks to stay in the low power mode for a longer period of cycles. It improves the power saving by 10%~15% on average. The rank level read-write reordering forces DRAM to handle read requests, which are critical to the system performance, as soon as it can. It reduces the system performance degradation caused by the power management without sacrificing much power saving. From the experiments, our work reduces the DRAM power consumption by around 75%, which is better than the previous work and the oracle solution. Meanwhile, it causes only 1%~3% system performance degradation, which is smaller than the existing power management policy.

As for the future, we will add a controller that dynamically adjust the throttle delay in runtime to relieve the system performance degradation. We will also explore the potential of combining the techniques proposed in this thesis with other related works such as the automatic data migration, which creates empty ranks that can be shut off until the throttle delay is reached. It is possible to implement our work on the hybrid main memories such as the cached DRAM, which decreases DRAM access and therefore reduces more power. Our work can also be improved by working with the write combining technique, which combines write requests at the last level cache. Furthermore, we will extend this work by utilizing the multiple low power modes provided by the most recent DRAM circuits.



# References

- [1] G. Zhang, H. Wang, X. Chen, S. Huang, and P. Li, "Heterogeneous multi-channel: Fine-grained DRAM control for both system performance and power efficiency," in *Proceedings of the 49th Design Automation Conference*, 2012.
- [2] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM design and organization for energy-constrained multi-cores," in *Proceedings of the 37th International Symposium on Computer Architecture*, 2010.
- [3] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM energy management using software and hardware directed power mode control," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.
- [4] K. Chandrasekar, B. Akesson, and K. G. W. Goossens, "Run-time power-down strategies for real-time SDRAM memory controllers." in *Proceedings of the 49th Design Automation Conference*, 2012.
- [5] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.
- [6] D. Meisner, B. Gold, and T. Wensich, "PowerNap: Eliminating server idle power," in *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.
- [7] K. T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. C. Lee, and M. Horowitz, "Rethinking DRAM power modes for energy proportionality," in *Proceedings of the 45th International Symposium on Microarchitecture*, 2012.
- [8] *JEDEC Standard: DDR2 SDRAM Specification*, JEDEC Solid State Technology Association, 2009.
- [9] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proceedings of the 35th International Symposium on Computer Architecture*, 2008.
- [10] J. Janzen, "Calculating memory system power for DDR SDRAM," *Designline*, vol. 10, no. 2, 2001.
- [11] *Calculating memory system power for DDR3*, Micron Technology, Inc, 2007.
- [12] *512Mb: x4, x8, x16 DDR2 SDRAM*, Micron Technology, Inc., 2012.
- [13] H. Park, S. Yoo, and S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," in *Proceedings of the 48th Design Automation Conference*, 2011.

- [14] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proceedings of the 46th Design Automation Conference*, 2009.
- [15] N. AbouGhazaleh, B. Childers, D. Mosse, and R. Melhem, "Energy conservation in memory hierarchies using power-aware cached-DRAM," in *Proceedings of the 23rd International Conference on Computer Design*, 2005.
- [16] H. Zheng, Z. Zhang, E. Gorbato, and Z. Zhu, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *Proceedings of the 40th International Symposium on Microarchitecture*, 2008.
- [17] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proceedings of the 39th International Symposium on Computer Architecture*, 2012.
- [18] V. D. L. Luz, M. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *Proceedings of the 39th Design Automation Conference*, 2002.
- [19] M. Pedram, "Power optimization and management in embedded systems," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2001.
- [20] G. Thomas, K. Chandrasekar, B. Akesson, B. Juurlink, and K. Goossens, "A predictor-based power-saving policy for DRAM memories," in *Proceedings of the 15th Conference on Digital System Design*, 2012.
- [21] I. Hur and C. Lin, "A comprehensive approach to DRAM power management," in *Proceedings of the 14th International Symposium on High-Performance Computer Architecture*, 2008.
- [22] K. Chandrasekar, B. Akesson, and K. Goossens, "Run-time power-down strategies for real-time SDRAM memory controllers," in *Proceedings of the 49th Design Automation Conference*, 2012.
- [23] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.
- [24] J. Mukundan and J. F. Martinez, "MORSE: Multi-objective reconfigurable self-optimizing memory scheduler," in *Proceedings of the 18th International Symposium on High-Performance Computer Architecture*, 2012.
- [25] H. Hanson and K. Rajamani, "What computer architects need to know about memory throttling," in *Workshop on Energy-Efficient Design*, 2010.
- [26] C. J. Lee, V. Narasiman, E. Ebrahimi, O. Mutlu, and Y. N. Patt, "DRAM-aware last-level cache writeback: Reducing write-caused interference in memory system," Tech. Rep., Apr. 2010.
- [27] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in

*Proceedings of the 16th International Symposium on High-Performance Computer Architecture*, 2010.

- [28] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, “Multi2Sim: A simulation framework for CPU-GPU computing,” in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012.
- [29] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “DRAMSim2: A cycle accurate memory system simulator,” *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, Jan.–Jun. 2011.
- [30] *Cortex-A9 MPCore technical reference manual*, ARM, 2012.
- [31] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.
- [32] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proceedings of the 22nd International Symposium on Computer Architecture*, 1995.

