Architecting Memory Systems for Emerging Technologies

by

Byoungchan Oh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Trevor N. Mudge, Co-Chair
Assistant Professor Ronald G. Dreslinski Jr., Co-Chair
Professor David Blaauw
Professor William R. Martin

Byoungchan Oh
bcoh@umich.edu
ORCID iD: 0000-0001-9612-2501

© Byoungchan Oh 2018
All Rights Reserved

To my family and all those who wished me well.

ACKNOWLEDGMENTS

I would like to thank my primary advisor, Professor Trevor Mudge, for his invaluable support and guidance. I appreciate everything he has done for me. No words can express my gratitude to him. Professor Ronald Dreslinski, my co-advisor, has perfectly complemented Trev. I also would like to thank my dissertation committee members, Professor David Blaauw and William Martin, for their wisdom and guidance in the final steps of my studies.

I am pleased to have met lifelong friends in TRON lab: Antony Gutierrez, Qi Zheng, Nilmini Aberaytne, Yajing Chen, Cao Gao, Jonathan Beaumont, Dong-Hyeon Park, Yiping Kang, and Johann Hauswald. I shall never forget the countless hours and moments together. I especially appreciate the kindness, consideration, and help of Nilmini during whole my Ph.D. study and life. I would like to thank all those who directly and indirectly have helped my studies: Doowon Lee, Yongkee Kwon, Jihyae Bae, Dong-Keun Kim, Professor Jeongseob Ahn, and Professor Nam Sung Kim.

Lastly and most importantly, I would like to thank my parents and brother. I am forever grateful for their unwavering faith and support. To my loves, Blue, Berry and Mijung, I could finish my studies thanks to you. I owe much of my successes to my wife and her devotion to my family.

Thank you.

TABLE OF CONTENTS

Dedication . . . . **ii**

Acknowledgments . . . . **iii**

List of Figures . . . . **vi**

List of Tables . . . . **viii**

Abstract . . . . **ix**

Chapter

1 Introduction **1**

1.1 Memory System 1
1.2 Memory Technology 2
1.3 Emerging Memory Technology 6
1.4 Dissertation Organization 8

2 STT-MRAM Architecture for Smart Activation and Sensing **9**

2.1 Introduction 10
2.2 Challenges in Architecting STT-MRAM 13
2.2.1 Large Sense Amps with High Power Consumption 13
2.2.2 Limitations with Shared Sense Amps 14
2.3 SMART Architecture 17
2.3.1 Re-architecting STT-MRAM 18
2.3.2 Benefits 20
2.3.3 Discussion 26
2.4 Device Modeling 28
2.4.1 Area Model 29
2.4.2 Timing Model 30
2.4.3 Energy Model 30
2.5 Evaluation 32
2.5.1 Evaluation Methodology 32
2.5.2 Performance 33
2.5.3 Energy 35
2.5.4 Sensitivity Analysis 38
2.6 Related Work 39

2.7 Summary 40

3 Improving Load Balancing for Memory Channels 41

3.1 Introduction 42
3.2 Background 44
3.2.1 Increasing Demand of Memory Capacity and Bandwidth 44
3.2.2 High Bandwidth Memory 44
3.3 Challenges in Many Channel Memory Systems 48
3.3.1 Imbalanced Channel Utilization 48
3.3.2 Implementation Challenges of Memory Controllers 50
3.4 Overview of the Proposed Design 52
3.4.1 Re-architecting Memory Controllers 53
3.4.2 Re-architecting HBM 58
3.4.3 Overhead 59
3.5 Evaluation 61
3.5.1 Methodology 61
3.5.2 Performance Analysis 62
3.5.3 Area Overhead 65
3.6 Related Work 66
3.7 Summary 67

4 Conclusion 69

Bibliography 71

LIST OF FIGURES

1.1 Von Neumann architecture. 1
1.2 Memory hierarchy. 2
1.3 DRAM scaling trend [1]. 3
1.4 Increasing throughput loss and power consumption by refresh [2]. 3
1.5 Impacts of the DRAM page size in STREAM benchmark [3]. 4
1.6 Increasing data rate from SDR to DDR4. 5
1.7 STT-MRAM cell structure and MTJ. 6
1.8 2.5D integration between HBM and GPU through a interposer. 7

2.1 Bank architecture and operation of DRAM (left) and conventional STT-MRAM (right). 15
2.2 SMART bank architecture and data/control flow. 18
2.3 Change of read and write accesses after making sensing operation triggered by a RD command. 19
2.4 SMART page mode and enhanced bank architecture. 20
2.5 Relation between page size and ACT energy. 21
2.6 Impact of tRRD on bank-level parallelism. 22
2.7 tRRD and tFAW for various page sizes. tRRD and tFAW for DRAM are obtained from a DDR3 datasheet [4]. 23
2.8 Comparison of the three repair schemes. 24
2.9 Change of row access cycle on row misses. 25
2.10 Row hit rate in various workloads. 34
2.11 Read latency profile of mix6 workload. 35
2.12 Performance improvement compared to DRAM. 36
2.13 Energy savings over DRAM. 36
2.14 Breakdown of average memory power. 37
2.15 Normalized performance improvement and energy saving of the conventional STT-MRAM (Conv-Delay) to the baseline DRAM with various page size. . . . 38
2.16 Normalized IPC and energy to DRAM with various configurations (average of all mix workloads). 39

3.1 GPU systems with GDDR5 and HBM. 45
3.2 3D structure of HBM and a simple example of TSV connection to DRAM dies. 46
3.3 Comparison between two memory organizations [5]. 47
3.4 Channel Utilization. 48

3.5 Simple diagram for work stealing. 49
3.6 Performance according to queue depth. 50
3.7 Schedulers of the memory controller in many channel memory systems [6]. . . 51
3.8 Hierarchical queue structure. 53
3.9 Limited scheduling. 55
3.10 Avoiding timing constraint. 56
3.11 HBM with crossbars. 60
3.12 Performance improvement after the migration. 63
3.13 Change of imbalanced memory service time. 63
3.14 MPKI and locality. 64
3.15 Performance according to different queue configuration. 65

LIST OF TABLES

2.1 Comparison with previous studies 27
2.2 Area comparison 29
2.3 Timing and latency comparison 30
2.4 Energy and current comparison 31
2.5 Default system configuration 33
2.6 Workloads for multi-core simulations 33

3.1 Configured System 61
3.2 Workload list 62
3.3 Estimated area 66

ABSTRACT

The advance of traditional dynamic random access memory (DRAM) technology has slowed down, while the capacity and performance needs of memory system have continued to increase. This is a result of increasing data volume from emerging applications, such as machine learning and big data analytics. In addition to such demands, increasing energy consumption is becoming a major constraint on the capabilities of computer systems. As a result, emerging non-volatile memories, for example, Spin Torque Transfer Magnetic RAM (STT-MRAM), and new memory interfaces, for example, High Bandwidth Memory (HBM), have been developed as an alternative. Thus far, most previous studies have retained a DRAM-like memory architecture and management policy. This preserves compatibility but hides the true benefits of those new memory technologies.

In this research, we proposed the co-design of memory architectures and their management policies for emerging technologies. First, we introduced a new memory architecture for an STT-MRAM main memory. In particular, we defined a new page mode operation for efficient activation and sensing. By fully exploiting the non-destructive nature of STT-MRAM, our design achieved higher performance, lower energy consumption, and a smaller area than the traditional designs. Second, we developed a cost-effective technique to improve load balancing for HBM memory channels. We showed that the proposed technique was capable of efficiently redistributing memory requests across multiple memory channels to improve the channel utilization, resulting in improved performance.

CHAPTER 1

Introduction

1.1 Memory System

A memory system is one of the core sub-systems in modern computer architectures. Modern computer architectures are based on Von Neumann architecture which mainly consists of memory, input/output, the arithmetic/logic unit, and the control unit as shown in Fig. 1.1 [7, 8]. The memory in Von Neumann architecture stores program data and instruction data. Thus, this architecture is sometimes called to a stored-program computer [9]. Because storing data to memory and loading data from memory are essential operations in this architecture, performance and energy consumption of the memory system significantly affect overall system performance and energy consumption [10–12]. Moreover, the importance of a memory system continuously is increasing with emerging applications and services which need more memory capacity and bandwidth [13–15] .

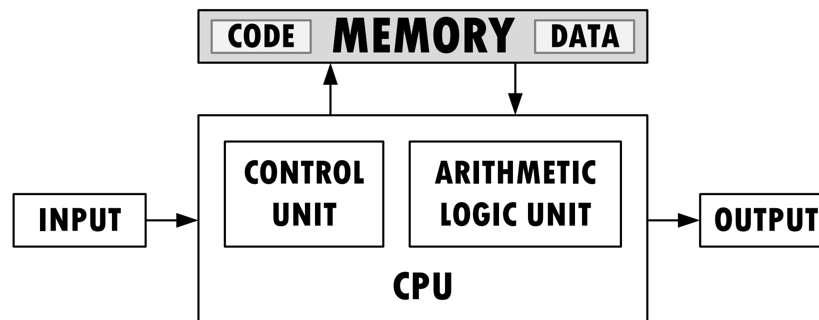


Figure 1.1: Von Neumann architecture.

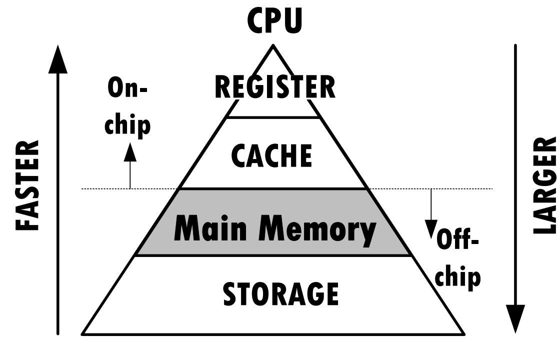


Figure 1.2: Memory hierarchy.

Memory systems have evolved with a hierarchical structure as shown in Fig. 1.2 [16]. This structure can take both advantages of short access time offered from small memories (*e.g*, register files and caches) and area efficiency provided from large memories (*e.g*, main memory and storage). A main memory is not too fast, but not too small. Because the different process technology from that of CPU is used for main memories, they are typically off-chip devices. Generally, the main memory holds whole program code and data whereas caches temporally hold partial code and data[1]. Thus, there are frequent accesses to the main memory in irregular applications, where cached data are not reused well. Besides, recent in-memory database techniques increase use of the main memory [14, 17].

1.2 Memory Technology

Memory Cell. Dynamic Random Access Memory (DRAM) is a type of random access semiconductor memory that stores each bit of data in a separate tiny capacitor. It is used as the main memory in most computing systems. As the *de facto* standard in the past several decades, DRAM has been continuously increased in capacity as shown in Fig. 1.3. However, after increasing for decades, the capacity of DRAM nowadays has now stopped at 8 Gb. The root cause resulting in this trend is DRAM's refresh operation. Because the cell capacitor is leaky, DRAM cells must be periodically refreshed to prevent data loss

[1]If a main memory space is not enough, storage can be used as the main memory by the page swap

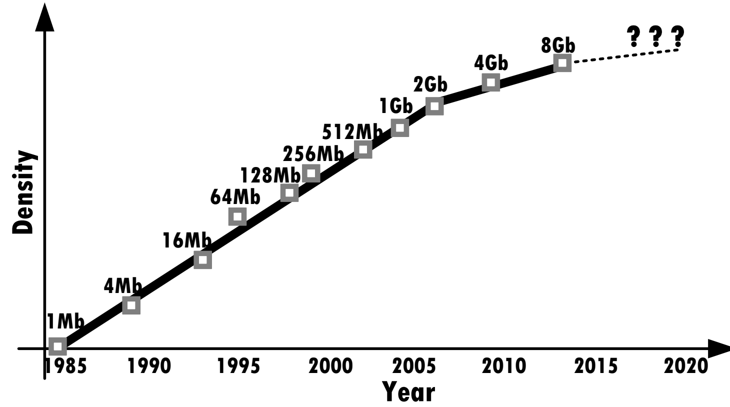
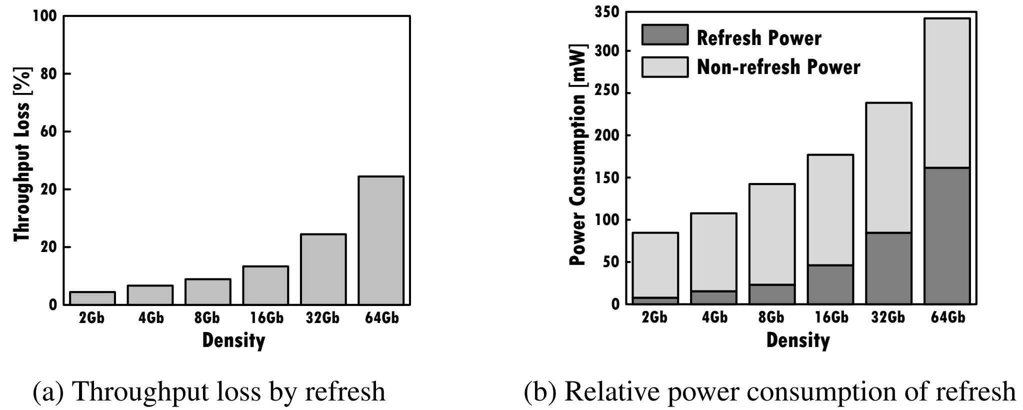


Figure 1.3: DRAM scaling trend [1].



(a) Throughput loss by refresh

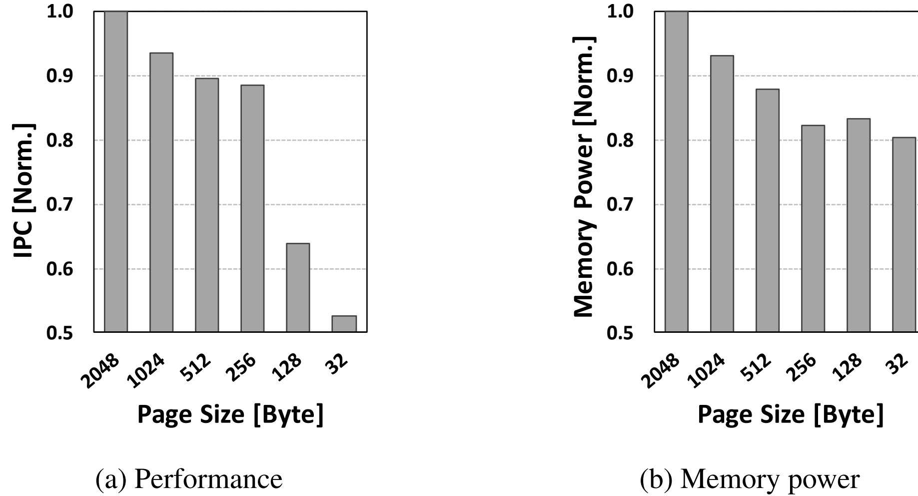
(b) Relative power consumption of refresh

Figure 1.4: Increasing throughput loss and power consumption by refresh [2].

within their retention time (typically 64 ms). Unfortunately, this retention time has not improved.[2] If DRAM capacity is increased without improvement of retention time, more DRAM cells must be refreshed within the same period. Thus, the relative time spent for refresh is increased according to the capacity. This increased refresh time can be converted to throughput loss as shown in Fig. 1.4a. Furthermore, the power consumption for refresh is also increased in proportion to capacity. The refresh power consumption, which does not contribute to computation, will become comparable with the dynamic power consumption, which is directly involved with the computation as shown in Fig. 1.4b.

Memory Interface. Double Date Rate (DDR), which is successors to Single Data Rate

[2]In fact, the retention time is becoming worse and thus some recent DRAM devices have shorter retention time (32 ms) than the normal 64 ms [18].



(a) Performance

(b) Memory power

Figure 1.5: Impacts of the DRAM page size in STREAM benchmark [3].

(SDR), is the most widely used interface standard for a main memory. In DDR memories, a memory access is divided into two types of accesses, row access and column access. A row access command activates or deactivates a row in a memory array. When a row is activated, all cells connected to that row are sensed by sense amps and the sensed data are stored in row buffers. Because data are already stored in the row buffer, a following column access command can be served from the row buffer without additional array access.[3] Also, a following column access going to the same row can be served without additional row access, because all data for the row are already stored in the row buffer. This operation is called *page mode* operation and still remains unchanged in modern DDR devices. The size of data sensed by a row activation command is defined as page size. In general, page mode increases performance by avoiding repeated activation of the same row to access different columns. Thus, the large page size is good for performance especially when data locality is high. However, the wasted energy is increased in large pages because all data in a page (*i.e*, the row buffer) are not used always. As shown in Fig. 1.5, it is hard to capture both high performance offered by large pages and low power consumption offered by small pages at the same time [19–22].

[3]A read command can immediately read out data stored in the row buffer, but a write command needs to access the array after updating data in the row buffer. However, after updating the row buffer but before finishing array access, another column access is allowed.

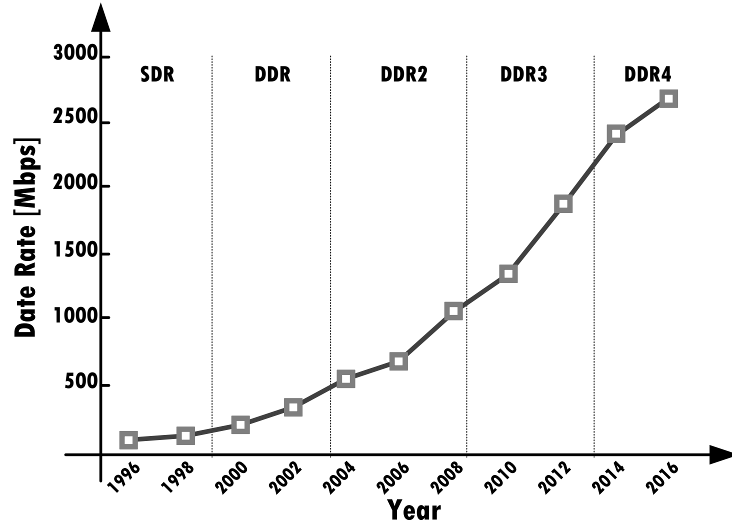
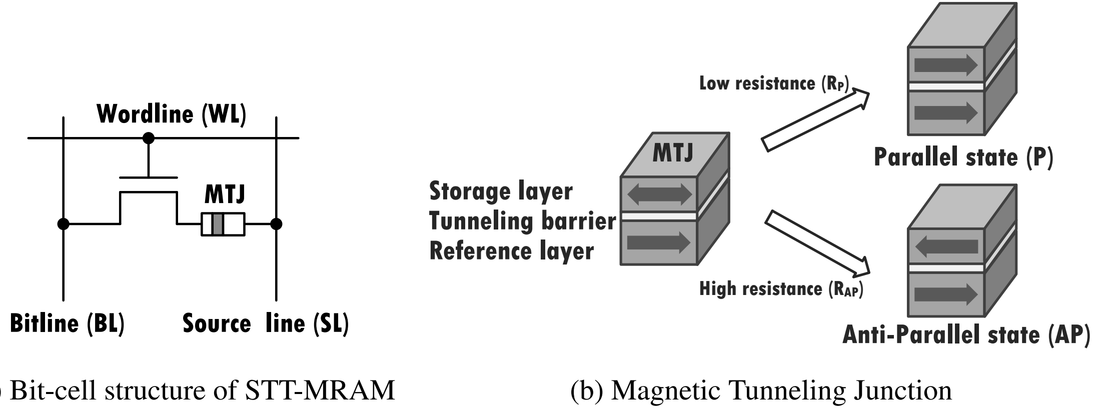


Figure 1.6: Increasing data rate from SDR to DDR4.

Total peak memory bandwidth is defined as,

$$Memory\ Bandwidth = Data\ Rate \times Bus\ Width \times Number\ of\ Channels \tag{1.1}$$

where data rate is equal to memory bus clock speed in SDR and double of the clock speed in DDR, bus width is traditionally is 64 bits, and the number of channels is typically 1∼6 for main memory systems. As shown in Fig. 1.6, data rate from SDR to DDR4 has been continuously increased. That is, traditionally memory bandwidth has been increased by increasing data rate (*i.e*, clock speed). However, increasing the clock speed has several challenges. First, the clock speed in latest DDR generation is already 1.2 GHz and it is fast enough. There will be some degree of increase in the clock speed, but that would not be an order of magnitude increase. That is, it is hard to achieve an order of magnitude higher memory bandwidth by increasing the clock speed. Second, generally faster clock requires higher supply voltage. Although technology improvements such as low voltage swing termination logic (LVSTL) suppress the increase of the supply voltage, increasing clock speed cannot save power consumption.



(a) Bit-cell structure of STT-MRAM

(b) Magnetic Tunneling Junction

Figure 1.7: STT-MRAM cell structure and MTJ.

1.3 Emerging Memory Technology

Memory Cells Technology. Because of DRAM's refresh and the limitations caused by it, emerging non-volatile memories such as Resistive RAM (ReRAM), Phase Change Memory (PCM) and Spin Transfer Torque Magnetic RAM (STT-MRAM) have been receiving more attention. However, unlike other emerging non-volatile memories, STT-MRAM is considered to be a replacement candidate for a main memory because of its fast access speed and high endurance [23–29]. An STT-MRAM cell consists of an access transistor and a Magnetic Tunneling Junction (MTJ) to store data as shown in Fig. 1.7a. There are three layers in an MTJ: two magnetic layers and one tunneling barrier between them as shown in Fig. 1.7b. The "reference" magnetic layer has a fixed magnetization and the "free/storage" layer has a variable magnetization. When their magnetization directions are the same (parallel state, P), an MTJ shows low electrical resistance (R_P). When their magnetization directions are different (anti-parallel state, AP), they show high resistance (R_{AP}). This change in the resistance is called the tunneling magnetoresistance effect. The spin transfer torque (STT) force is exerted by injecting a spin-polarized current to change the magnetization direction of the free/storage layer. In order to enable bi-directional change, a bi-directional current should flow through an STT-MRAM cell. Therefore, one terminal of an MTJ is connected to a bit-line (BL) through a selection transistor and the other terminal is connected to a

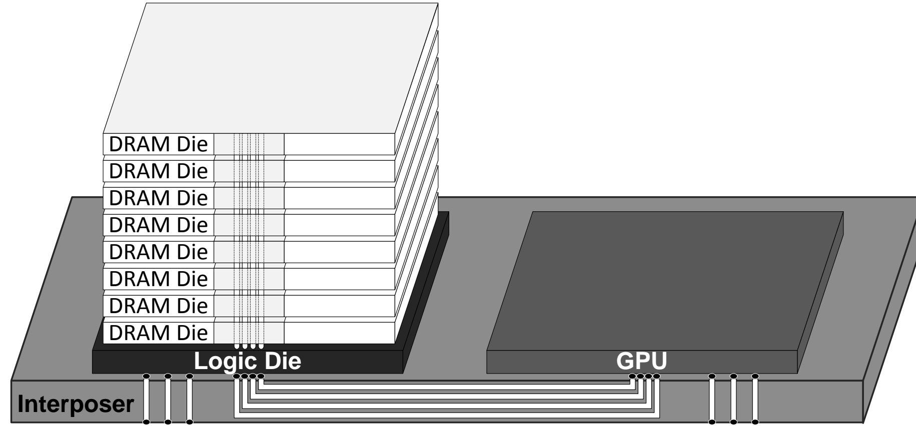


Figure 1.8: 2.5D integration between HBM and GPU through a interposer.

source-line (SL) for the write operation as shown in Fig. 1.7a [30, 31]. Although this cell structure is similar to a DRAM cell structure, the STT-MRAM cell is larger due to the larger access transistor. The larger access transistor is required to drive enough current to switch the magnetization direction. Alternatively, increasing the write time (pulse width) can reduce the amount of current required and thus the cell size can be smaller [32, 33].

Memory Interface Technology. There are variants of DDR standard for specialized applications: Low Power DDR (LPDDR) optimized for power consumption and Graphic DDR (GDDR) optimized for performance. Although several unique features exist in LPDDR or GDDR such as Partial Array Self Refresh (PASR) in LPDDR and separate higher speed Write Clock (WCK) in GDDR, their basic operations are almost same as normal DRAM [5, 18,34]. Besides, they all use narrow (32∼64 bits per channel) and fast (up to 7Gbps per pin) memory channels. Because of this narrow and fast channel, increasing memory capacity and bandwidth is limited by the power budget. Moreover, accommodating more memory chips to achieve higher the capacity and bandwidth can limit the form factor. In order to overcome this issues, 3D stacked DRAM memories, such as High Bandwidth Memory (HBM) and Wide I/O (WIO), have been developed [35,36]. In Fig. 1.8, an example of 2.5D integration between a processor (GPU) and a stacked DRAM (HBM) through a silicon interposer is described. Because the interconnections through the interposer are much shorter

than the interconnections between off chips on PCB, the data transfer energy is lower on the interposer. In addition, a stack of DRAM dies in HBM provides multiple channels (up to 8) within a small space and the interposer can accommodate all I/Os for the channels. Because the bandwidth is linearly proportional to the data rate and total number of I/Os for all channels, HBM's large number of I/Os (1024 vs. 32 for GDDR5) allows to lower the data rate (2Gbps vs. 7Gbps for GDDR5) while increasing the bandwidth (256GB/s vs. 28GB/s for GDDR5).

1.4 Dissertation Organization

The research work is organized as follows. Chapter 2 presents a co-design of memory architecture and its management policy for an STT-MRAM main memory. Chapter 3 presents a cost-effective technique to improve load balancing for HBM channels. Finally, Chapter. 4 includes the summary of this work and conclusions.

CHAPTER 2

STT-MRAM Architecture for Smart Activation and Sensing

STT-MRAM is a promising memory technology as a drop-in replacement of DRAM for main memory, because it can offer higher energy efficiency than DRAM with latency comparable to DRAM. However, STT-MRAM needs to employ current sense amps which consume an order of magnitude larger space and power than voltage sense amps adopted by DRAM. Consequently, to manage the high cost of sense amps, STT-MRAM decouples bit-lines from sense amps and shares one sense amp with 16$\sim$128$\times$ bit-lines, exploiting the non-destructive nature of its read operation. However, such STT-MRAM reduces the size of row buffers and incurs more row-buffer misses (*i.e*, higher activation energy and lower performance) than DRAM along with other issues, especially when it follows interfaces and policies designed for DRAM.

To cost-effectively address these issues, we propose SMART, co-designing STT-MRAM architecture and its management policy in this study. Specifically, unlike DRAM and conventional STT-MRAM, SMART proposes to sense bit-lines after receiving a column access command instead of a row activation command. This can provide several benefits including larger pages, fewer sense amps, lower activation/sensing power, shorter latency, fewer address pins and more efficient repairs of defective columns than conventional STT-MRAM. Our evaluation shows that SMART consumes 11% (39%) lower energy while providing 9% (5%) higher performance than conventional STT-MRAM (DRAM) on average. In addition

to these benefits, SMART is 6% smaller than conventional STT-MRAM.

2.1 Introduction

STT-MRAM is one of the promising emerging non-volatile memory (NVM) technologies as a drop-in replacement of DRAM for main memory because of its faster speed and higher endurance than other NVM technologies [23–29]. However, STT-MRAM has some disadvantages over DRAM. One of such disadvantages is a need to use large and high-power sense amps in STT-MRAM. Specifically, STT-MRAM needs current sense amps consuming an order of magnitude larger space and higher power than voltage sense amps adopted by DRAM. To manage the cost of implementing such sense amps, STT-MRAM leverages the non-destructive nature of its read operation and shares one sense amp with 16∼128 bit-lines in each bank [37–42]. Such STT-MRAM architectures, nonetheless, suffer from two limitations.

First, STT-MRAM with fewer sense amps provides smaller row buffers and thus pages than DRAM, as the number of sense amps determines the size of row buffers [16]. Larger pages provide higher performance with more row-buffer hits when data locality is high, but they consume more energy when data locality is poor, which is also known as the overfetching problem. In contrast, smaller pages consume less energy when data locality is poor, but they give lower performance with more row-buffer misses when data locality is good. The size of STT-MRAM row buffers is far smaller than that of DRAM row buffers but much larger than that of a column access. Therefore, STT-MRAM suffers from more row-buffer misses than DRAM without completely eliminating the overfetching problem.

Second, when such STT-MRAM uses interfaces and policies designed for DRAM, it suffers from a column address fragmentation problem [27, 42]. Specifically, DRAM requires sense amps to sense all 16,384 bit-lines in a sub-array and latch the data until open-

ing another row due to the destructive nature of its read operation. Therefore, for a given row address, DRAM performs both activation and sensing at the same time with a single row activation (ACT) command. However, it becomes problematic to directly apply such an activation/sensing approach to STT-MRAM with fewer sense amps than the number of bit-lines (*i.e*, size of a row), because STT-MRAM must receive both a row address and some of a column address at the same time to connect a sub-set of chosen bit-lines to the sense amps through multiplexers when activating a row. That is, STT-MRAM with the same capacity as DRAM requires more address pins to send not only a row address but also some of a column address together with a row command[1]. Besides, such a fragmentation of a column address between row and column commands considerably worsens efficiency and flexibility of column repair mechanisms as a row or column command has only partial column address information [43, 44].

To cost-effectively address these issues, we propose SMART, **S**TT-**M**RAM **AR**chi**T**ecture supporting smart activation/sensing. Specifically, exploiting one of the advantages of STT-MRAM, non-destructive nature of its read operation, we propose to provide only 64 sense amps[2] for each bank and make a column access command instead of a row activation command sense bit-lines. This deceptively simple change, which is not possible for DRAM with the destructive nature of its read operation, can offer the following advantages over conventional STT-MRAM.

(1) SMART offers the illusion of providing 16× larger row buffers with 16× fewer sense amps than conventional STT-MRAM. That is, SMART provides 2KB pages with only 64 sense amps per bank, whereas conventional STT-MRAM gives 128B pages with 1,024 sense amps per bank. Furthermore, conventional STT-MRAM repeatedly consumes long time (tRC = 27.5ns) and high power to access the same row but columns which

[1] DRAM uses the same set of address pins to receive both row and columns addresses in a time multiplexed manner. Since a row address typically needs more bits than a column address, the number of row address bits determines the number of address pins in DRAM.

[2] In this study, we assume DRAM and STT-MRAM modules, each consisting of eight ×8 devices for 64-bit I/O. Therefore, each device must sense 64 bits to support the burst length of 8 for a single column access command.

were not selected and sensed by the previous row command. This is because conventional STT-MRAM needs to recognize such memory accesses as row-buffer misses to select and sense different bit-lines. In contrast, SMART can recognize such memory accesses as row-buffer hits and it only needs another column access command because it selects and senses different bit-lines which were already connected to the necessary cells by the previous row activation command.

(2) SMART consumes ∼88% lower activation power than conventional STT-MRAM. Specifically, conventional STT-MRAM senses 1,024 bit-lines as part of a row activation, and sensing power dominates the activation power. In contrast, SMART consumes 16× less sensing power than conventional STT-MRAM because it senses only 64 bit-lines (*i.e*, granularity of a column access) as part of a column access. Hence, SMART practically eliminates the overfetching problem. Furthermore, whenever a memory write access demands an activation of another row, conventional STT-MRAM unnecessarily consumes sensing power because sensing is part of the row activation. However, SMART does not consume any sensing power for such a memory write access, because sensing is not part of a row activation but part of a column read access.

(3) SMART offers shorter latency for memory accesses. Specifically, high sensing power imposes `tRRD` and `tFAW` constraints in DRAM and STT-MRAM and limits the number of row activation commands in a certain time period. As SMART significantly reduces sensing power, it can eliminates these two constraints and handle more row activation commands in a shorter time period than conventional STT-MRAM. As previously mentioned, conventional STT-MRAM performs sensing as part of row activation while sensing is unnecessary for memory write accesses. Moreover, sensing constitutes a notable fraction of activation time. Therefore, SMART can also reduce latency of memory write accesses especially to different rows compared with conventional STT-MRAM.

(4) SMART needs fewer pins and offer 10.7× more efficient repairs of defective columns than conventional STT-MRAM. Particularly, SMART does not sense bit-lines as part of

row activation. Therefore, it does not need to select and connect specific bit-lines to sense amps as part of row activation. That is, it only needs to receive a row address for activation like DRAM. Besides, SMART receives the full column address with a column command, it can use the same efficient mechanism as DRAM for repairing defective bit-lines.

(5) SMART eliminates the need for sending separate pre-charge commands since a column command can comprise the function of a pre-charge command. As a result, SMART not only offers 11% shorter latency for memory accesses which are directed to the same bank but incur row-buffer misses, but also consumes less command bus bandwidth than conventional STT-MRAM.

In summary, (1)–(5) not only reduce energy but also improve performance. Our evaluation shows that SMART consumes 11% (39%) lower energy while providing 9% (5%) higher performance than conventional STT-MRAM (DRAM) on average. In addition to these benefits, SMART is 6% smaller than conventional STT-MRAM.

2.2 Challenges in Architecting STT-MRAM

In this section, we compare STT-MRAM with DRAM and explicate challenges and limitations in conventional STT-MRAM.

2.2.1 Large Sense Amps with High Power Consumption

STT-MRAM is expected to offer cell read/write speed comparable to DRAM and good endurance ($> 10^{15}$) [45–49]. A recent study demonstrated 4Gb LPDDR2-compatible STT-MRAM with $9F^2$ cell size and sub-$50ns$ read/write speed [27]. Such characteristics make STT-MRAM a promising alternative to $\sim 7F^2$ DRAM. However, STT-MRAM poses unique challenges especially in implementing a sense amps (SA) [37, 50–54].

First, STT-MRAM needs a large current SA with a reference current generator because sensing small difference in on/off resistance is challenging, which is further worsened by

process variations. This requires STT-MRAM to adopt a very complex current SA which consumes an order of magnitude larger space than a voltage SA employed by DRAM.

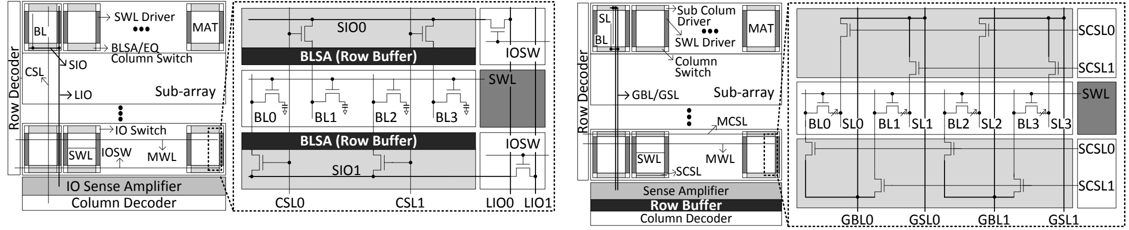
Second, sensing in STT-MRAM consumes high power. DRAM SAs simply charge or discharge bit-lines (BLs) once per sensing, whereas STT-MRAM SAs need to continuously flow current to BLs until they reach a level sufficient for sensing. To eliminate the power consumed by the continuous current flow, separate buffers are implemented and data in SAs are copied to the buffers so that the SAs can be turned off immediately after sensing bit-lines [41, 50].

2.2.2 Limitations with Shared Sense Amps

DRAM needs to directly couple every BL with a dedicated SA in each sub-array, as shown in Fig. 2.1a due to its destructive nature of read operation. STT-MRAM, however, shares one SA with 16∼128 BLs after it decouples SAs from BLs with multiplexers [27,37,38,42], as depicted in Fig. 2.1b. This is to manage the cost of SAs, exploiting its non-destructive nature of its read operation.

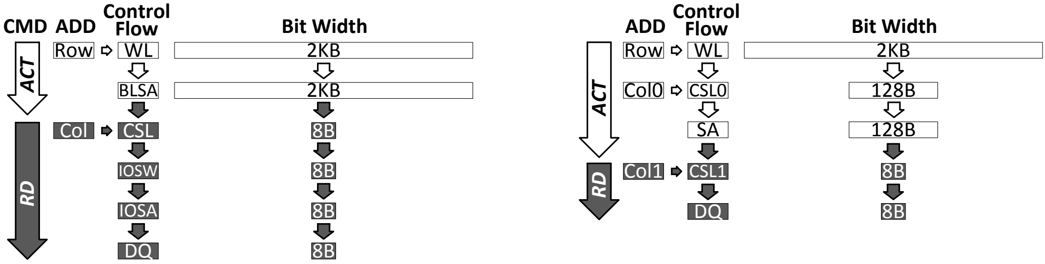
DRAM operating in page mode senses cell states of a row and stores them into a row buffer[3] when an `ACT` command is received. This allows DRAM to pre-charge/activate a row once for multiple column accesses, as depicted in Fig. 2.1c. Such an operating mode has not changed in modern DRAM architectures [34, 55–57]. Fig. 2.1d describes the steps of serving a read request in STT-MRAM following the same page mode as DRAM but sharing one SA with 16 BLs to reduce the cost. A row activation command first asserts a word-line (WL) connected to 16,384 cells (2KB). Since there are only 1,024 SAs (*i.e*, 128B row buffer), a column selection signal (CSL0) uses part of the column address (Col0) to select one BL out of 16 BLs. The column selection signal (MCSL-SCSL) in an STT-MRAM bank shown in Fig. 2.1b corresponds to the WL selection signal (MWL-SWL). The remaining part of the column address (Col1) selects global bit-lines (GBLs) and it is

[3] The 16,384 SAs shared by two adjacent sub-arrays serve as a row buffer.



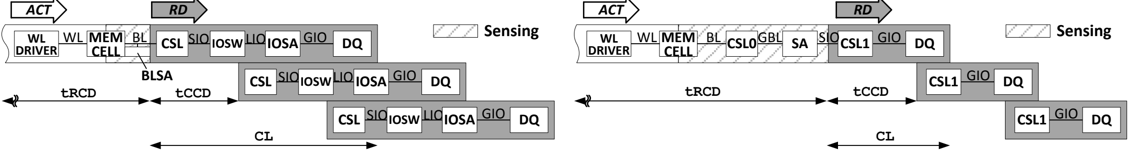
(a) Bank architecture and interconnect of DRAM

(b) Bank architecture and interconnect of conventional STT-MRAM



(c) Data/control flows for a single read operation in DRAM

(d) Data/control flows for a single read operation in STT-MRAM



(e) Page mode and pipelined RD operations in DRAM

(f) Page mode and pipelined RD operations in STT-MRAM

Figure 2.1: Bank architecture and operation of DRAM (left) and conventional STT-MRAM (right).

determined by a column access command (RD or WR). Such STT-MRAM, however, suffers from the following limitations.

Limitation 1: Longer latency. DRAM places bit-line sense amps (BLSAs) [25, 27] above and below a sub-array because the charge sharing limits the BL length in DRAM [58, 59]. Because the local I/O (LIO) lines are too long to be driven by small BLSAs, larger I/O sense amps (IOSAs) are placed near the column decoder of each bank to assist the transfer of data through the LIO lines. In contrast, STT-MRAM does not require BLSAs [25, 27], placing only column multiplexers above and below a sub-array with one set of SAs at the bottom of a bank consisting of 128 sub-arrays. The column multiplexers connect one BL in a group of BLs to a SA through a GBL, which makes the sensing path of STT-MRAM

longer than that of DRAM. That is, the sensing path of STT-MRAM is the height of a bank while that of DRAM is the height of a sub-array. This in turn increases `tRCD`, minimum time from `ACT` to `RD` or `WR` while decreasing `CL` as shown in Fig. 2.1f [42].

Limitation 2: Smaller pages. Consider STT-MRAM sharing one SA with N BLs where N is 16∼128 in prior work [27, 37, 60]. When STT-MRAM has 16,384 cells connected to a WL like DRAM, the page size of such STT-MRAM becomes 1/N of that of DRAM. Although not all data in pages are always used, smaller pages degrade performance of applications especially with high data locality such as streaming. Furthermore, with fewer SAs than BLs (*i.e*, cells in a row), STT-MRAM encounters the following three cases: (1) an access to another row (*i.e*, row miss); (2) an access to the same row with BLs selected and sensed by previous `ACT` (*i.e*, row hit); or (3) an access to the same row but BLs which are not selected and sensed by previous `ACT` yet (*i.e*, row hit but row buffer miss). The third case needs to be handled like a row miss because connecting appropriate BLs to SAs and sensing them are coupled with `ACT` in STT-MRAM sharing one SA with many BLs. That is, some of a column address becomes part of a row address since they are needed before sensing appropriate BLs. Consequently, selecting different BLs which were not selected by previous `ACT` always demands another `ACT`.

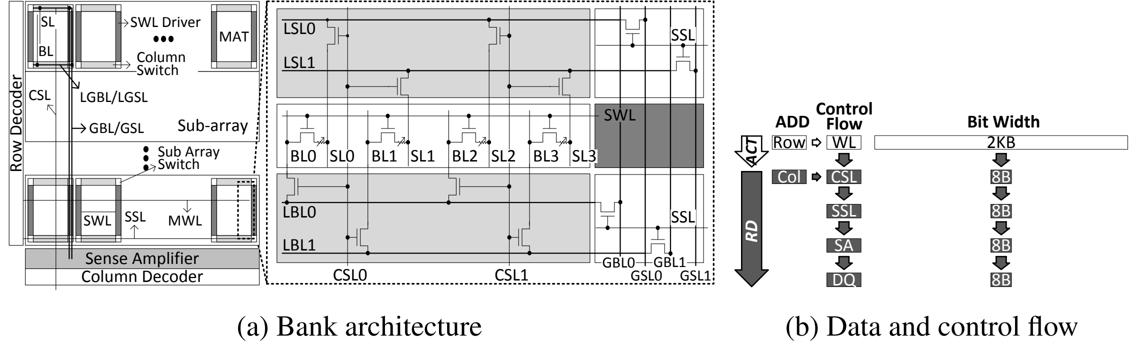
Limitation 3: Lower repair efficiency. The fact that a column address is split between row activation and column access commands, *referred to as column address fragmentation in this study*, creates two important challenges in managing chip yield and compatibility with existing DDR interfaces. Typically, the minimum repair granularity is a row or a column. For the sake of repair efficiency, any redundant WL can replace any defective WL in the same bank and any redundant BL can replace any faulty BL in the same mat. This technique is known as any-to-any replacement [61]. In STT-MRAM architecture sharing one SA with multiple BLs, however, both `ACT` and `RD`/`WR` do not have the full column address information. STT-MRAM uses the partial column address from `ACT` to select one BL in every BL group. That is, it selects one BL from each BL group and each mat has 64

BL groups. Therefore, if the partial column address to replace a defective BL in a BL group, it needs to replace every BL selected by the same partial column address. Consequently, STT-MRAM needs at least one redundant BL for every BL group (*i.e*, 64 redundant BLs per mat). On the other hand, STT-MRAM uses the partial column address from `RD` or `WR` to select 64 BL groups from 1,024 BL groups (*i.e*, the number of SAs). Hence, if it uses the partial column address to replace a defective BL in a BL group, it needs to replace every BL in the BL group. That is, STT-MRAM needs at least one redundant BL group in each mat, (*i.e*, 16 BLs per group). Both cases negatively affect chip yield because they limit the flexibility of any-to-any replacement and increasing the minimum repair granularity.

Limitation 4: Higher pin cost. In DRAM, row and column addresses share the same address pins because they are delivered at different times [34, 56]. The number of address pins is equal to the number of row address bits because there are typically more rows than columns (*e.g*, 64K rows and 2K columns in $\times$8-8Gb DRAM). In STT-MRAM, however, more address pins are needed to send some of a column address with a row address. For example, we need 4 more pins for STT-MRAM with N = 16. Therefore, STT-MRAM with the shared SA architecture is not compatible with conventional DDR interfaces. To solve this problem, comboAS [27, 42] proposes to start actual row activation only after `RD` or `WR` sends a column address, but it always increases `CL` by the row activation time. LPDDR2-NVM [62, 63] proposes another command, `PREACT` to deliver a part of the row address before sending `ACT` so that STT-MRAM can compose a complete row address after receiving the `ACT` command. This also increases row activation time while consuming more command bus bandwidth.

2.3 SMART Architecture

Due to the destructive nature of read operation, DRAM requires an `ACT` command to sense the state of every cell in a row after the WL is asserted. This demands the number of SAs to



(a) Bank architecture

(b) Data and control flow

Figure 2.2: SMART bank architecture and data/control flow.

be equal to the number of cells in a row. Therefore, the number of cells in a row determines both the page size and the activation power in DRAM. This makes it impossible for DRAM to offer both high performance of large pages and low power of small pages at the same time. In contrast to DRAM, because of the non-destructive nature of read operation, STT-MRAM does not need to make an `ACT` command sense the state of every cell in a row after asserting the WL. Exploiting such a property, we propose to re-architect STT-MRAM which senses BLs as part of `RD` instead of `ACT` and re-define its page mode operation. In the remainder of this section, we first present the detail of SMART architecture and then discuss the five key benefits of SMART over conventional STT-MRAM and/or DRAM.

2.3.1 Re-architecting STT-MRAM

In SMART a given `ACT` command completes its operation immediately after asserting a WL. That is, the `ACT` command does not sense any BL but it is a subsequent `RD` command that senses 64 BLs specified by the column address of the `RD` command. This, in turn, allows SMART to provide 2KB pages with only 64 SAs per bank along with other significant benefits which will be discussed in Sec. 2.3.2. To efficiently support such `ACT` and `RD` commands for SMART, we propose a bank architecture depicted in Fig. 2.2a.

Traditionally, the I/O interconnect of DRAM has a hierarchy as depicted in Fig 2.1a. Specifically, the interconnect from a memory cell to a SA is called BL. For a given column access, 64 BLs/BLSAs are selected by column selection lines (CSLs) and connected to

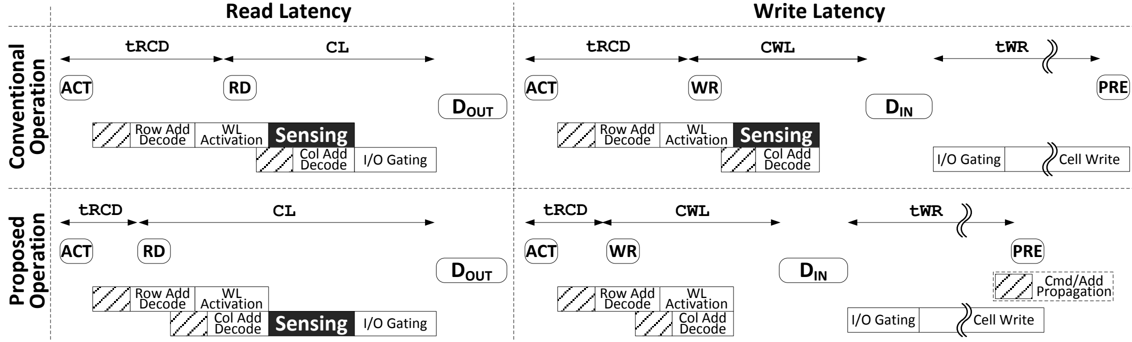
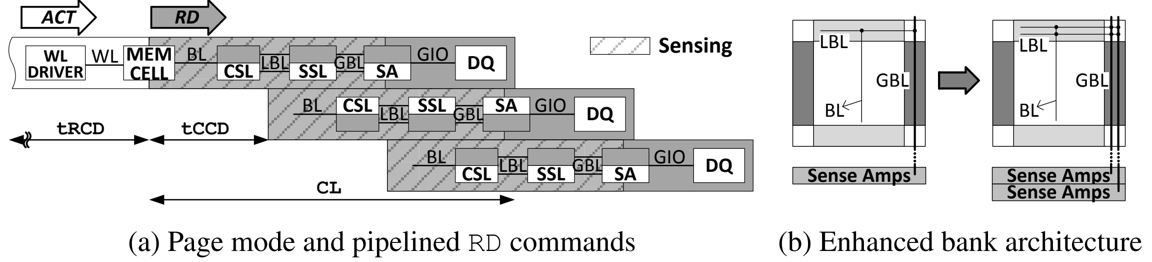


Figure 2.3: Change of read and write accesses after making sensing operation triggered by a RD command.

segmented I/O (SIO) lines in a sub-array through column switches (*i.e*, multiplexers). Then, the SIO lines are connected to local I/O (LIO) lines running vertically across a bank through I/O switches (IOSWs) [64]. The SIO, LIO and IOSW in DRAM correspond to local BL (LBL), GBL, and sub-array selection line (SSL), respectively, in SMART. Lastly, 64 SAs are placed at the bottom of each bank where IOSAs (*cf*, Sec. 2.2.2) are located in DRAM. In summary, SMART has a bank architecture similar to DRAM, but SMART has only 64 SAs per bank whereas DRAM has 16,384 SAs per sub-array.

This SMART bank architecture does not increase latency of memory read accesses, because the total amount of time for performing ACT and RD remains the same as conventional STT-MRAM. As shown in Fig. 2.3(left), compared with conventional STT-MRAM, SMART simply reduces the amount of time for ACT (tRCD) while increasing the amount of time for RD (CL). On the other hand, as shown in Fig. 2.3(right), SMART can decrease latency of memory write accesses. Specifically, compared with STT-MRAM, ACT-WR does not consume any time for sensing BLs without affecting CWL, time between the moment at which a WR command is sent and the moment at which its first data (D_{IN}) is placed.

In DRAM, the SIO-LIO lines are electrically isolated from the GIO lines, as shown in Fig. 2.1a. This allows DRAM to internally pipeline more than two consecutive RD commands which can be issued at every tCCD interval (typically ∼5ns) without waiting for long CL (typically ∼13.75ns), as shown in Fig. 2.1e. SMART, however, has longer



(a) Page mode and pipelined RD commands (b) Enhanced bank architecture

Figure 2.4: SMART page mode and enhanced bank architecture.

read time than tCCD because RD also senses BLs, as shown in Fig. 2.4a. To hide such long sensing time for consecutive RD commands, we propose to double the number of LBL and GBL lines, column multiplexers, and SAs, as shown in Fig. 2.4b. The two sensing paths take turns to serve consecutive RD commands so that SMART can handle one RD command at every tCCD interval, as depicted in Fig. 2.4a. Our analysis in Sec. 2.4.1 shows that the cost of doubling the number of sensing paths does not increase the cost of STT-MRAM. In fact, SMART decreases the cost by ∼6% because SMART still uses 8× fewer SAs than conventional STT-MRAM.

2.3.2 Benefits

With the bank architecture presented in Sec. 2.3.1, SMART can provide the following notable benefits over conventional STT-MRAM.

Benefit 1: Larger pages and fewer SAs. SMART with a re-defined page mode can give the illusion of providing larger pages but demanding fewer SAs than conventional STT-MRAM. Specifically, an ACT command of SMART simply asserts a WL to connect every cell in the row to 16,384 BLs, and it is a subsequent RD command that selects appropriate 64 BLs and senses them. That is, SMART needs only 64 SAs per bank for 2KB pages whereas conventional STT-MRAM implements 1,024 SAs per bank for 128B pages. Because conventional STT-MRAM implements 16×fewer SAs than BLs, it repeatedly consumes long time (tRC = 27.5ns) for accessing the same row but columns which were not selected and sensed by the previous ACT command. SMART, however, does not consume tRC for such

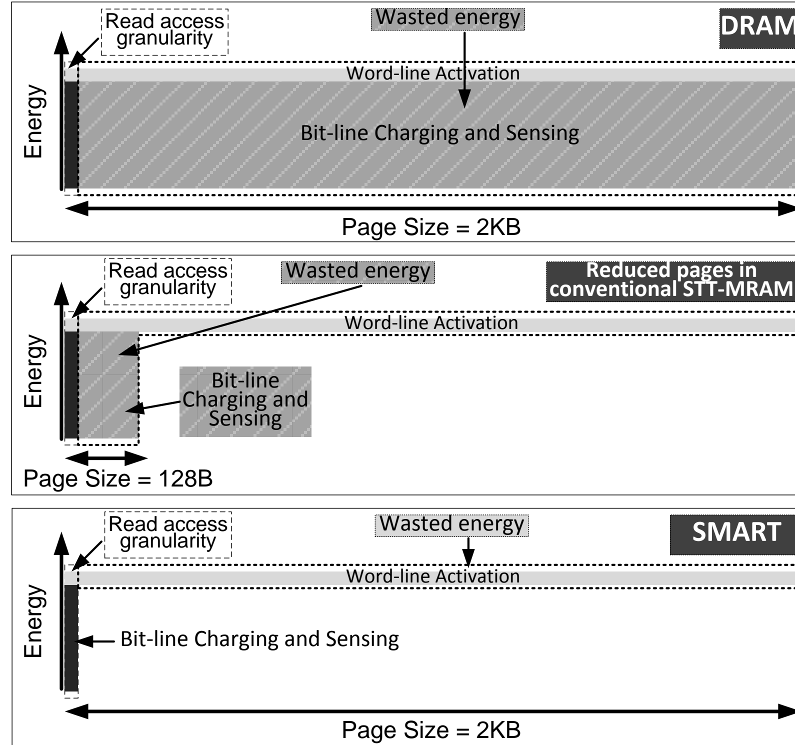


Figure 2.5: Relation between page size and ACT energy.

column accesses because it only needs another RD command to connect and sense these columns. This can significantly reduce the latency of sequential memory accesses. Note that the SAs in conventional STT-MRAM consume ∼9% of STT-MRAM space based on an adapted version of DRAMSpec [65]. That is, SMART can reduce the space and activation power consumed by SAs to 12.5% of conventional STT-MRAM. This is sufficient not only to negate the cost increased by another read path but also to significantly reduce sensing power and memory access latency, as elaborated below.

Benefit 2: Lower activation power with fewer SAs. ACT energy is a major contributor to total DRAM energy in DRAM. To reduce the ACT energy, DRAM architectures with smaller pages and fine-grained activation have been proposed [19–22]. In contrast, SMART reduces ACT energy while providing larger pages with fine-grained activation at a smaller cost than STT-DRAM. Fig. 2.5 illustrates the relationship between page size and ACT energy in DRAM, STT-MRAM and SMART. In a DRAM device providing 2KB pages, ACT

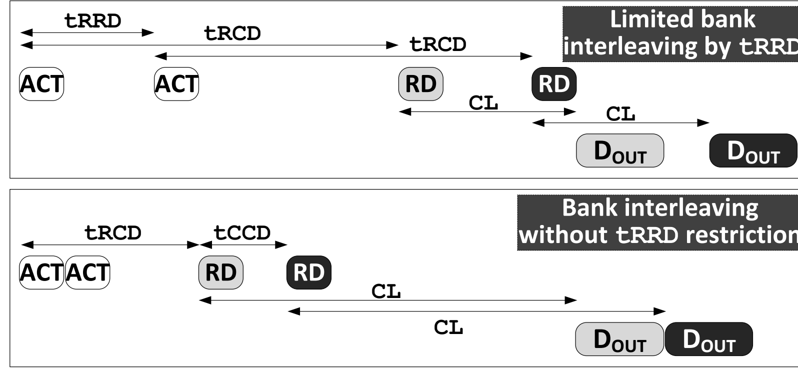
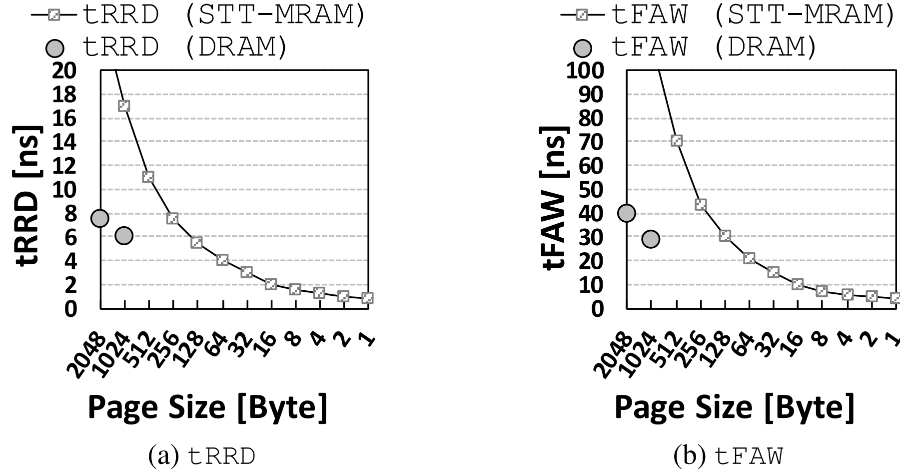


Figure 2.6: Impact of `tRRD` on bank-level parallelism.

asserts a WL, connecting 16,384 cells to 16,384 BLs. Although we consider higher WL voltage (V_{PP}) than the BL voltage (V_{DD}) and low efficiency of the charge pump to generate V_{PP} ($\sim$30% [66]), we see that charging/discharging the BLs still dominates the `ACT` energy because of the large number of BLs. Since a `RD` command accesses only $\sim$0.4% (= 64/16,384) of cells in a row, sensing 16,384 BLs for only a few `RD` accesses, referred to as the overfetching problem, wastes a significant amount of energy in DRAM, as shown in Fig. 2.5(top).

Benefit 3: Shorter latency with lower activation power. High `ACT` power affects not only total memory energy but also overall memory performance. Specifically, simultaneous activation of multiple rows, each charging/discharging 16,384 BLs, draws a large amount of current. This requires some time to recover from the voltage drop of the power delivery network, which is enforced by `tRRD` (RAS to RAS delay) and `tFAW` (four activation window). If there are two read accesses to different banks, the second `ACT` command can be scheduled between the first `ACT` and `RD` commands, but not until `tRRD` has elapsed from the first `ACT` command, as depicted in Fig. 2.6(top). `tFAW` limits the number of `ACT` commands to four within a `tFAW` time window. Therefore, these constraints limit bank-level parallelism and they are imposed on not only DRAM but also other memory technologies. For example, LPDDR2-NVM also defines `tRRD` and `tFAW` [63].

We plot `tRRD` and `tFAW` for various page sizes in Fig. 2.7, where the maximum activation current values are determined by a method of prior work [67]. This shows that `tRRD`



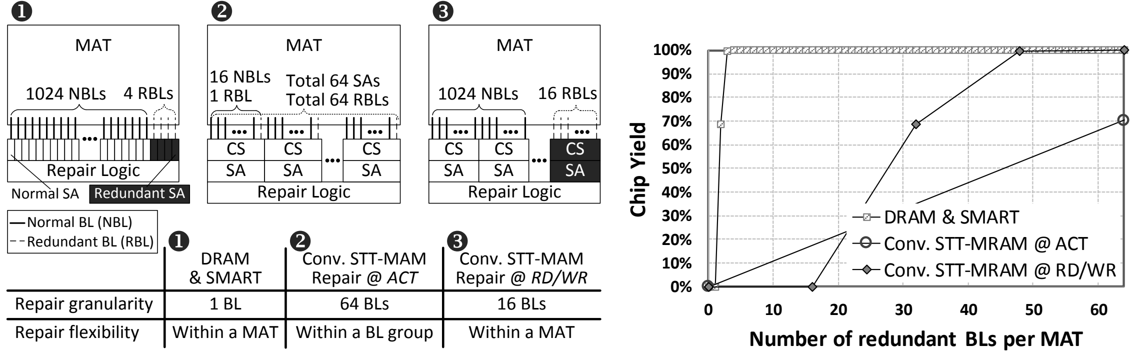
(a) `tRRD` (b) `tFAW`

Figure 2.7: `tRRD` and `tFAW` for various page sizes. `tRRD` and `tFAW` for DRAM are obtained from a DDR3 datasheet [4].

and `tFAW` of conventional STT-MRAM with the page size of 128B are $\sim$6.2ns and $\sim$30ns, respectively. Since a current SA consumes far more power than a voltage SA, STT-MRAM needs longer `tRRD` than DRAM for the same page size. In contrast, SMART activates 16$\times$ fewer SAs and thus consumes less power than conventional STT-MRAM. Note that SMART `ACT` does not consume any sensing power and the recovery time for activating 64 SAs is short enough compared to `tCCD` ($\sim$5ns). Hence, SMART can practically eliminate these two constraints and handle multiple `ACT–RD` commands to different banks back to back. This significantly reduces the latency of memory accesses especially for memory-intensive applications. For example, SMART can serve two `ACT–RD` commands to two different banks without consuming `tRRD`, as shown in Fig. 2.6(bottom).

Lastly, SMART also consume shorter time and less power than conventional STT-MRAM for memory write accesses. This is because an `ACT` command of SMART does not consume time and power for sensing which is unnecessary for memory write accesses, whereas conventional STT-MRAM still does.

Benefit 4: Fewer pins and more efficient repair. Unlike conventional STT-MRAM, SMART does not demand any part of a column address with `ACT`, and thus it needs the same number of address pins as DRAM. That is, SMART does not suffer from the column address fragmentation problem discussed in Sec. 2.2.2. As discussed in Sec. 2.2.2, repair-



(a) Example of various column repair schemes (b) Chip yield according to repair schemes

Figure 2.8: Comparison of the three repair schemes.

ing a mat of conventional STT-MRAM is not as efficient and flexible as DRAM because the column address fragmentation problem splits a column address between ACT and RD. This can significantly increase the cost of repairing mats or decrease the yield of STT-MRAM chips. Fig. 2.8a describes the column repair schemes for DRAM and conventional STT-MRAM. In DRAM (❶), we can replace any 1,024 BLs with any 4 redundant BLs in a mat, and we repair a BL with a RD or WR command comprising a complete column address [68]. As SMART also exposes a complete column address to a RD or WR command, it can adopt the same column repair scheme as DRAM. In conventional STT-MRAM, however, neither a ACT command nor a RD/WR command has a complete column address. This makes repairing BLs far less efficient and flexible than DRAM or SMART.

Consider STT-MRAM with 1,024 BLs per mat and N (= 16) BLs per SA, (*i.e*, 1,024/N = 64 BL groups). If we are to repair a BL with ACT (❷), we need one redundant BL for every N BLs (= 64 redundant BLs) as ACT can select only one BL in each BL group. On the other hand, if we are to repair a BL with RD or WR (❸), we need to replace the entire BL group including a defective BL with a redundant BL group (= 16 redundant BLs) because RD or WR can select only BL groups.

We analyze the chip yield for various numbers of redundant BLs in Fig. 2.8b where we assumed that capacity of a chip is 8Gb, the number of mats is 16,384 in a chip where each

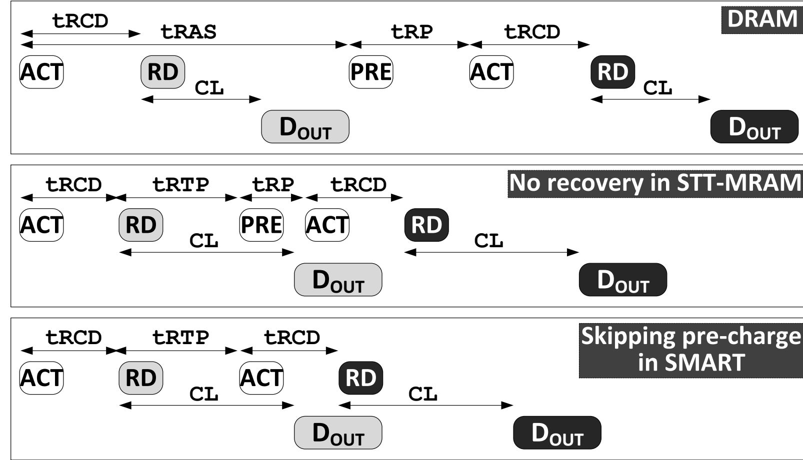


Figure 2.9: Change of row access cycle on row misses.

mat has 512 WLs and 1024 BLs, N is 16 for conventional STT-MRAM, memory defects follow a `Poisson` distribution [68, 69], the target bit error rate (BER) after row repair is 10^{-7} and both DRAM/SMART and conventional STT-MRAM use the same row repair scheme. This shows that conventional STT-MRAM has a lower chip yield than DRAM or SMART for the same number of redundant BLs. In other words, to accomplish the same chip yield (99%) under the same BER (10^{7}), conventional STT-MRAM requires 10.7× more redundant BLs than DRAM and SMART.

Benefit 5: Eliminating precharge commands. To activate another row in the same bank (handle a row-buffer miss), a precharge (`PRE`) command needs to be sent to DRAM before an `ACT` command. Specifically, `PRE` in DRAM consists of two phases: (1) deactivating an asserted WL and (2) initializing BLs before `ACT` senses BLs. (2) also destroys cell states if it is performed before the WL is completely deactivated. Therefore, (1) should be completely done before (2) is started, and the amount of time for (1) and (2) (`tRP`) increases latency of memory accesses when row-buffer misses occur, as shown in Fig. 2.9(top). Furthermore, when STT-MRAM follows the same page mode operation as DRAM, it experiences more row-buffer misses with smaller pages and thus it pays this penalty more frequently than DRAM.

Unlike DRAM, however, STT-MRAM does not need to sequentially perform (1) and

(2) because of the non-destructive nature of its read operation. Moreover, STT-MRAM immediately transfers the cell states sensed by SAs to registers serving as a row buffer (Sec. 2.2.1). This allows STT-MRAM to initialize the BLs and SAs immediately after sensing BLs as part of `ACT` and reduce `tRP`. Note that STT-MRAM initializes its BLs by discharging them to V_{SS}. The driving strength of charging BLs is limited to prevent unexpected changes of cell states (read disturbance) [40, 70], but that of discharging BL is not limited. Hence, the amount of time for (2) can be much shorter than DRAM and it is already included in `tRAS` instead of `tRP` [27, 42, 50]. This reduces `tRP` of STT-MRAM by the amount of time for (2). Consequently, as shown in Fig. 2.9(middle), STT-MRAM can serve the second `RD` command faster than DRAM, but it does not reduce or hide the amount of time for (1), still demanding a separate `PRE` command. In contrast, SMART can overlap the amount of time for (1) (~3.7ns) with the amount of time to decode a given row address and compare the address with addresses in a row repair table [68] during the early phase of `ACT` for the next row (~4.2ns)[4]. This allows SMART to completely remove `PRE` right before `ACT`, further reducing the latency to handle row-buffer misses. Note that prior work only reduces `tRP` [42,59] while SMART does not consume `tRP` at all to handle row-buffer misses.

2.3.3 Discussion

SMART can consume far less space for SAs and redundant BLs than conventional STT-MRAM but more space for another sensing path per bank. Overall, SMART is ~6% smaller than conventional STT-MRAM (Sec. 2.4.1). We summarize the key differences among DRAM, LPDDR2-NVM, conventional STT-MRAM and SMART in Table 2.1

SMART does not increase the latency of serving a single read request or consecutive read requests issued at the `tCCD` interval but it still increases CL. This may increase overall

[4] In this case, we leverage the ~0.5ns difference, but some overlap between deactivating the previous WL and activating the current WL is still acceptable because such overlap does not destroy the cell states.

Table 2.1: Comparison with previous studies

	DRAM	LPDDR2-NVM [38,63]	Conv-Delay [27,42]	Conv-Pin [24,41]	This work (SMART)
Page size	Large (2KB)	Small (32B)	Small (128B)	Small (128B)	Large (2KB)
Activation energy	High	Medium	Medium	Medium	Extremely low
Bank-level parallelism	Limited by tRRD/tFAW	Same as DRAM	Same as DRAM	Same as DRAM	No limitation
Pin-compatible with DDR	Yes	Yes (3-phase addressing)	Yes (delayed ACT)	No	Yes
Repair flexibility	High	Low	Low	Low	High
PRE technique	SALP [59]: Can violate tRP to access different sub-array	No	EarlyPA [42]: Internally perform PRE after the sensing and set tRP=1	No	No PRE command and no tRP
Need software/OS support	No	Yes, for write operation	No	No	No

Highlighted green: good features, highlighted yellow: good, but limited, (Conv-Delay and Conv-Pin are the conventional STT-MRAM designs)

read latency of serving multiple read requests issued at longer intervals than `tCCD`. However, our evaluation shows that the performance degradation caused by the increased `CL` is outweighed by the performance increase by larger pages, higher bank-level parallelism, and lower row-buffer miss latency.

Since a row buffer holds data only for a previously accessed column, SMART cannot compare-before-write when a `WR` command is sent to a different column of the activated row. Comparing data before writing reduces write energy and improves the endurance by not overwriting the same data [27, 71]. However, the high endurance of STT-MRAM cells ($> 10^{15}$) guarantees practically unlimited write operations (Sec. 2.2.1). Moreover, cell write energy is not a major component in overall write energy. Therefore, such a technique has limited impact on giga-bit scale STT-MRAM (Sec. 2.4.3).

Lastly, as this work focuses on re-architecting STT-MRAM for higher performance and lower energy, we do not discuss challenges related to its cells, such as thermal stability, write endurance, and read disturbance in detail [47, 70]. Nonetheless, recent studies have demonstrated small (sub-20nm) STT-MRAM cells that can offer fast switching time (sub-10ns) under low write current (sub-10uA), high write endurance ($> 10^{15}$), thermal stability and read disturbance [72–74].

2.4 Device Modeling

To evaluate DRAM, STT-MRAM and SMART, we take DRAMSpec [65], a detailed timing, power, and area exploration tool which is originally developed for DRAM but can be adapted for other memory technologies such as STT-MRAM. For the baseline DRAM, we consider $\times 8$ 8Gb DRAM devices. While keeping the same chip floor-plan as DRAM, we adapt the bank architecture and interconnect models to model STT-MRAM and SMART with parameters taken from NVSim [75] and prior work [27, 28, 32, 33, 46] and then extrapolated to 30nm technology.

Table 2.2: Area comparison

	DRAM	Conventional STT-MRAM	SMART
Cell size	7.2F^2	10.9F^2	10.9F^2
SA size	1,213F^2	27,111F^2	27,111F^2
# of SA per bank	1,048,576	1,024	128
Bank area (μm^2)	2,914×7,512	3,132×8,695	2,980×8,390
Chip area (mm^2)	185.65	224.19 (21% ↑)	211.48 (14% ↑)

2.4.1 Area Model

We assume a 7.2F^2 DRAM cell (2F×3.6F = WL×BL) provided by the DRAMSpec's 30nm technology model and a 10.9F^2 (3F×3.6F) STT-MRAM cell. For conventional STT-MRAM, we take N = 16 which is from an industry STT-MRAM chip [27]. Following the JEDEC standard for DRAM, SMART has the same number of rows and columns as DRAM. However, SMART can implement any page size with the number of SAs equal to the number of bits per column access.

We also assume 12 redundant WLs and BLs per mat for DRAM and SMART (default in DRAMSpec) whereas we suppose 32 redundant BLs per mat for conventional STT-MRAM (Sec. 2.3.2). Prior work [50] demonstrated the layout area of various SAs for STT-MRAM, but it designed the SAs with a logic technology. Thus, we convert transistor sizes and design rules to those of a memory technology based on the ITRS roadmap to re-estimate the area [76].

We summarize the analyzed area of key memory components in Table 2.2. The total height of the BLSA blocks in DRAM is ∼20% of the total chip height in a 20nm 8Gb DRAM device [77]. Therefore, the number and size of SAs greatly affect the total chip size. SMART has 2× more sensing paths than conventional STT-MRAM, but it consumes ∼6% smaller space because it needs 8× fewer SAs. Albeit SMART uses 1.5× and 21.4× larger cells and SAs than DRAM, it uses 8,192× fewer SAs. This is because that a bank has 128 sub-arrays and a pair of two sub-arrays shares 16,384 SA in DRAM. SMART is

only 14% larger than DRAM whereas conventional STT-MRAM is 21% larger.

2.4.2 Timing Model

Table. 2.3 summarizes the timing parameters and read access latency of DRAM, conventional STT-MRAM and SMART, based on the memory clock frequency of 800MHz. DRAM gives the shortest latency for a single read access because the sensing speed of DRAM is faster than that of STT-MRAM. However, the overall latency of multiple read accesses is not determined not only by the sensing speed but also by other timing parameters such as `tRRD`, `tFAW`, `tRAS` and `tRP`, especially when different banks and rows need to be accessed.

Table 2.3: Timing and latency comparison

	DRAM	Conventional STT-MRAM	SMART
`tRCD` (clock cycle)	11	17 (1)	8
`tRAS` (cc)	27	18 (19)	9
`tWR` (cc)	12	19 (19)	19
`tRP` (cc)	11	4 (4)	4
`tRTP` (cc)	6	1 (18)	9
`tRRD` (cc)	6	5 (5)	1
`tFAW` (cc)	32	24 (24)	4
`CL` (cc)	11	8 (25)	17
Latency for single read	22	25 (26)	25
Latency for five reads - all different banks	54	49 (50)	41
Latency for two reads - same bank, but different rows	60	47 (48)	42

The numbers in ( ) are for the delaying `ACT` like comboAS [27,42].

2.4.3 Energy Model

In conventional STT-MRAM, activating fewer SAs reduces the energy consumption of a single `ACT` command. SMART, however, completely removes sensing energy from `ACT`

and thus it gives much smaller ACT energy than conventional STT-MRAM, as shown in Table 2.4. Instead, SMART includes the sensing energy in RD and thus the RD energy is higher than DRAM and conventional STT-MRAM. The conventional STT-MRAM has the lowest RD energy because of the reduced read path. Besides, STT-MRAM consumes higher WR energy than DRAM because of higher write current per cell. However, considering the energy consumption to transfer data across the chip through long interconnects, the impact of the cell write energy on the total write energy is limited.

Table 2.4: Energy and current comparison

	DRAM	Conventional STT-MRAM	SMART
ACT (nJ)	1.28	0.45	0.09
Single RD (nJ)	0.27	0.26	0.31
Single WR (nJ)	0.28	0.35	0.34
Energy for 8B read (nJ)	1.54	0.71	0.39
Energy for 2KB read (nJ)	69.99	86.56	78.19
Energy for 8B write (nJ)	1.55	0.81	0.43
Energy for 2KB write (nJ)	72.79	96.80	87.40
IDD0 (mA)	67	64	43
IDD2P (mA)	14	17	16
IDD2N (mA)	36	39	38
IDD3N (mA)	51	39	38
IDD4R (mA)	122	121	127
IDD4W (mA)	122	132	131
IDD5 (mA)	245	-	-

Table 2.4 shows the dynamic energy consumption of a single memory device for a single column (8B) request and a single page (2KB) request. If all columns in a page are accessed, STT-MRAM consumes more energy because of its inherent higher sensing and cell write energies. However, a single read/write access in STT-MRAM consumes less energy than DRAM because of the low ACT energy consumption. Comparing the two STT-MRAM's energies, SMART is more energy-efficient in all cases. This is because there is no wasted energy for the single column access and there are fewer ACT commands for the 2KB access. In addition, because SMART does not include sensing energy in write

requests, the gap between the two STT-MRAM designs in a 2KB write is larger than that in a 2KB read.

IDD2P is the power-down current and it is close to the sum of total transistor leakage. Thus, IDD2P is usually proportional to the total transistor width under the same technology. For simplicity, we increase IDD2P of STT-MRAM linearly with chip size. IDD2N and IDD3N are background current under precharge-standby and active-standby, respectively. The difference between IDD2P and IDD2N mainly results from the address/clock buffer current components. Thus, we assume the same increment for STT-MRAM's IDD2N. However, IDD3N stems from DRAM's unique leakage component. When a row in a bank is activated, 32,768 BLs are fully charged or discharged, whereas all BLs are precharged to V_{DD}/2 level when the row is deactivated. If the BL length is 512, then 16,777,216 cells are connected to the 32,768 BLs. The increased voltage difference between BL to a cell transistor increases leakage current, which is mostly GIDL and junction leakage current [78]. Because of this large number, small leakage current changes cause huge increases in IDD3N. However, the BL/SL condition of STT-MRAM is different from that of DRAM during active-standby, because BLs and SLs are always discharged except during sensing and writing. Therefore, we assume IDD3N is the same as IDD2N in STT-MRAM.

2.5 Evaluation

2.5.1 Evaluation Methodology

We evaluate SMART using MARSSx86 [79] and DRAMSim2 [80]. The configured system for the evaluation is shown in Table 2.5. Power-down mode is enabled to minimize standby power when there are no pending requests in the memory controller.

We employ two benchmark suites: SPEC2006 [82] and STREAM [3]. For multi-core simulations, multi-program workloads are composed as shown in Table 2.6. Misses-per-

Table 2.5: Default system configuration

Component	Specification
Processor	single and quad core
Last Level Cache	2MB-8 way (single), 4MB-16 way (quad)
Memory Controller	FR-FCFS [81], open-page, Ch:Ra:Ro:Ba:Co
Memory System	8GB single rank/ch (8Gb x8-DDR3L-1600)

kilo-instructions (MPKI) increases from mix1 to mix9. For all workloads, one billion instructions are simulated in their region of interest. In our evaluation, there are two conventional STT-MRAM designs with the shared SA structure: Conv-Pin and Conv-Delay. Conv-Pin emulates the designs proposed in [24, 41]. However, they have no consideration for the expanded address pins which result from the shared SA structure and are not compatible with JEDEC DDR. On the other hand, Conv-Delay internally delays the activation instead of increasing pin count [27, 42].

Table 2.6: Workloads for multi-core simulations

Workload	Application list
mix1	bzip, povray, astar, libquantum
mix2	hmmer, sjeng, xalancbmk, libquantum
mix3	gobmk, h264ref, astar, lbm
mix4	povray, omnetpp, soplex, lbm
mix5	sjeng, xalancbmk, mcf, stream_scale
mix6	h264ref, povray, hmmer, lbm
mix7	bzip, omnetpp, stream_copy, stream_scale
mix8	gobmk, milc, stream_add, stream_copy
mix9	hmmer, mcf, stream_triad, stream_add

2.5.2 Performance

In SMART, we strive to implement large pages (2KB) with low cost. Memory performance is affected by row buffer hit rate and page size. Both DRAM and SMART can implement 2KB pages, while the conventional STT-MRAM designs can implement only 128B pages. The row buffer hit rates are shown in Fig. 2.10. SMART has a row hit rate within 1% of

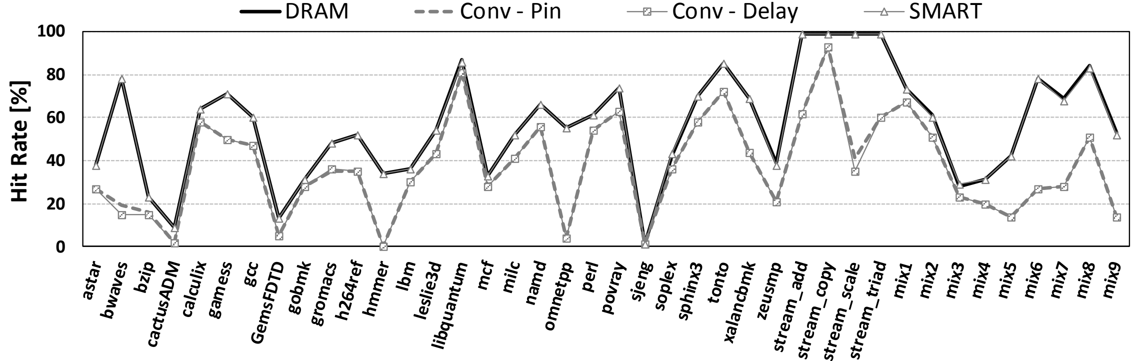


Figure 2.10: Row hit rate in various workloads.

DRAM's. However, both conventional designs have significantly lower hit rates, especially in STREAM workloads. Although all designs show low hit rate in a few workloads, the designs with the larger pages still perform better.

Fig. 2.11 shows the overall read latency distribution of the three devices under the mix6 workload. DRAM has a long tail latency because of refresh, but STT-MRAM has a short tail. In the Conv-Pin, we clearly observe two peaks in its distribution corresponding to read latency under row hits and misses. In SMART, there is no clear second peak because row misses are serviced faster and their read latency is partially overlapped with read latency under row hits. Although SMART has no data in 10∼19 cc range due to its long `CL`, it mostly falls within 30∼39 cc, implying that it is neither too quick nor too slow. DRAM also has most of its latency within 30∼39 cc, but its long tail negatively affects the overall latency. The average read latency of DRAM, Conv-Pin, and SMART is 52.6, 57.9 and 47.1 cc, respectively.

Fig. 2.12 shows system IPC improvement over DRAM. Conv-Pin and Conv-Delay degrade IPC on average by 3.7% and 4.3%. For some memory intensive workloads IPC degrades more than 30%. The biggest drawback in conventional STT-MRAM designs is the small page size. A substantial drop in row hit rate as compared to DRAM (*e.g*, bwaves, sphinx3 and STREAMs) significantly degrades their IPC. For workloads with similar row hit rate (*e.g*, lbm and milc), IPC is better than DRAM due to the removed refresh and

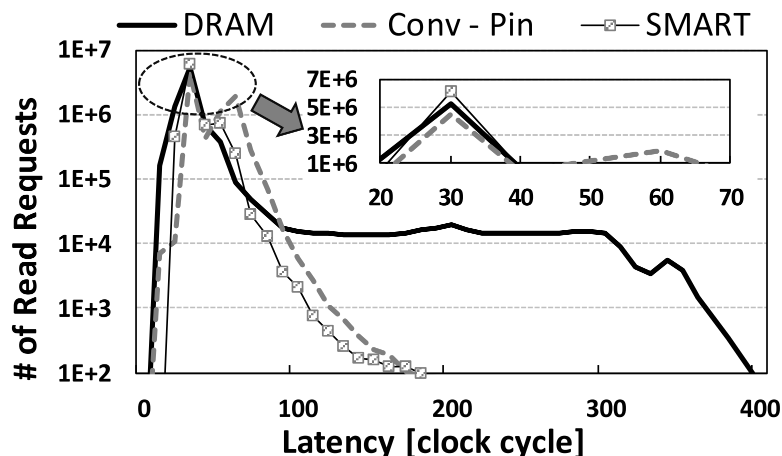


Figure 2.11: Read latency profile of mix6 workload.

restoration operations and the reduced precharge time. For non-memory-intensive workloads, regardless of hit rate, the IPC difference is negligible. Between the two conventional designs, Conv-Delay shows worse overall performance than the Conv-Pin because of its long `CL`.

SMART, on the other hand, improves IPC on average by 5.1% over DRAM. Because SMART has the same page size as DRAM, it achieves row hit rates as high as DRAM for applications having sequential memory accesses. In addition, SMART improves performance for applications having random memory accesses (low row hit rate) due to better row-miss latency and bank-level parallelism. As a result, MPKI is correlated to the IPC difference. In memory intensive workloads with (MPKI $>$15) (*e.g*, GemsFDTD, lbm and libquantum), there is up to a 34% IPC improvement over DRAM. Non-memory intensive workloads with (MPKI $<$ 1) (*e.g*, gamess, and namd), show no significant IPC improvement.

2.5.3 Energy

SMART has three advantages in energy over DRAM. First, `ACT` energy is extremely low, because sensing, which was the main energy contributor to `ACT`, was moved to `RD`. Second, cell leakage current is eliminated while the bank is activated (low active-standby power).

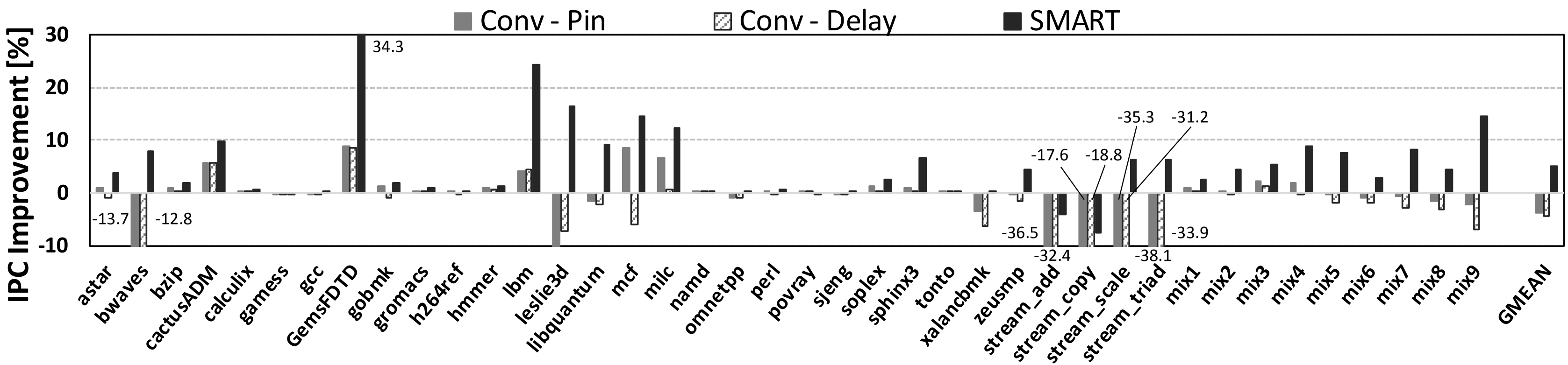


Figure 2.12: Performance improvement compared to DRAM.

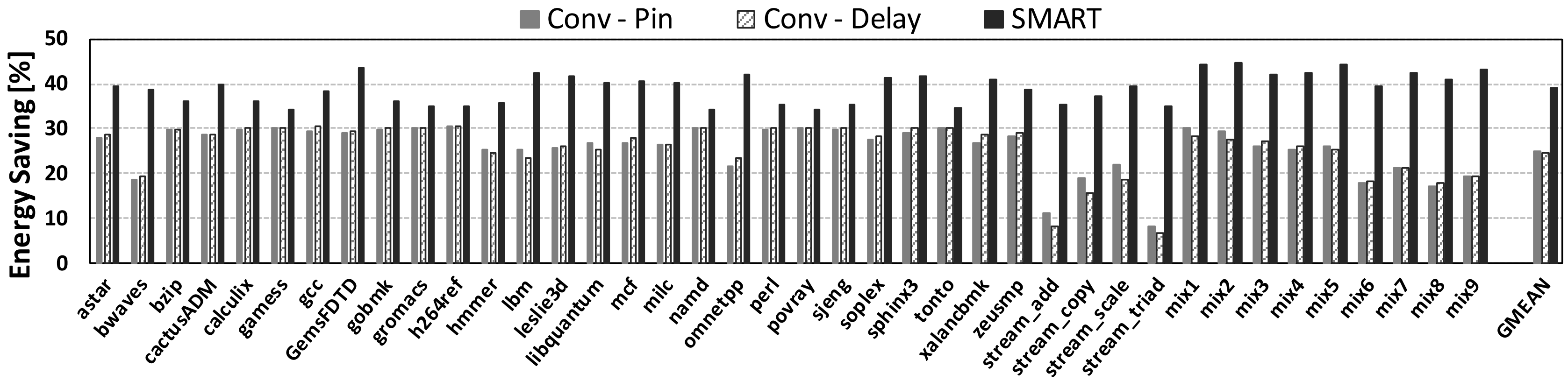


Figure 2.13: Energy savings over DRAM.

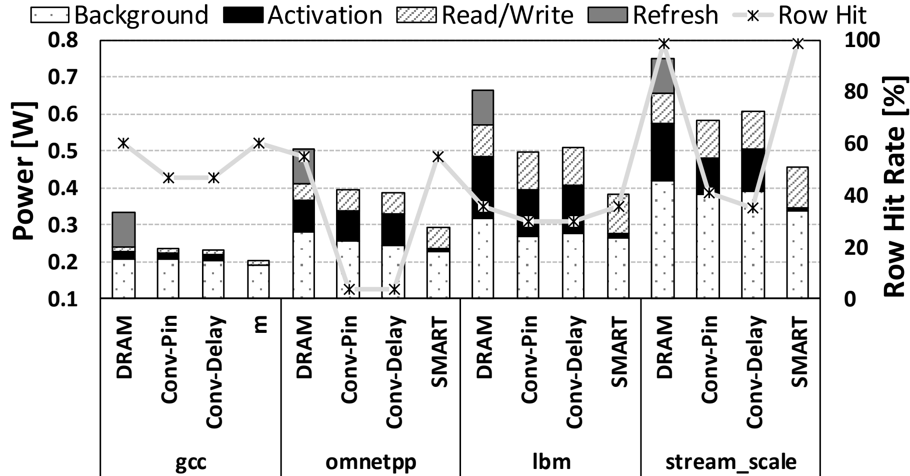
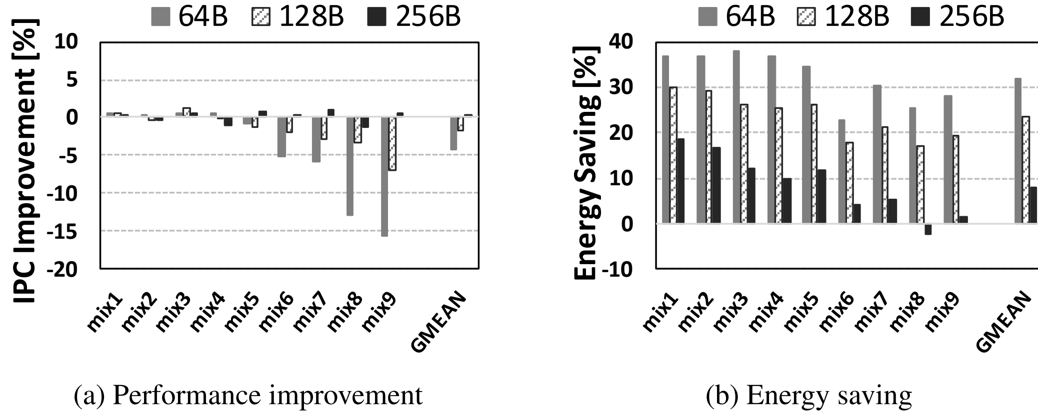


Figure 2.14: Breakdown of average memory power.

Last, STT-MRAM cells are non-volatile and have no refresh. While both the conventional STT-MRAM designs enjoy similar advantages over DRAM, they saved ACT energy by reducing the page size and impacted the row hits.

Fig. 2.14 breaks down average memory power. ACT power is higher than RD/WR power in DRAM. The average refresh power is 14∼29% of the total memory average power and its relative portion increases in non-memory intensive workloads (*e.g*, gcc). Although ACT power in the conventional STT-MRAM is less than in DRAM, the difference decreases when the conventional designs have low row hit rates (*e.g*, omnetpp and stream_scale). In these workloads, because memory has fewer chances to enter power-down mode due to row misses, the background power becomes higher than in DRAM in spite of the lower I_{DD3N}. RD/WR power is also higher than DRAM because of higher cell write current. In contrast, in SMART, the ACT power does not dominate total memory power. Although its RD/WR power is the highest among the four devices, the total dynamic power, which is the sum of ACT and RD/WR power, is the lowest. In addition, because SMART enters power-down mode as often as DRAM, background power stays low.

Fig. 2.13 shows the energy savings over DRAM. On average, the Conv-Pin and Conv-Delay save 24.9% and 24.5% of energy and SMART saves 38.9%. Due to small page size, energy savings of the conventional STT-MRAM is more sensitive to the row hit rate



(a) Performance improvement

(b) Energy saving

Figure 2.15: Normalized performance improvement and energy saving of the conventional STT-MRAM (Conv-Delay) to the baseline DRAM with various page size.

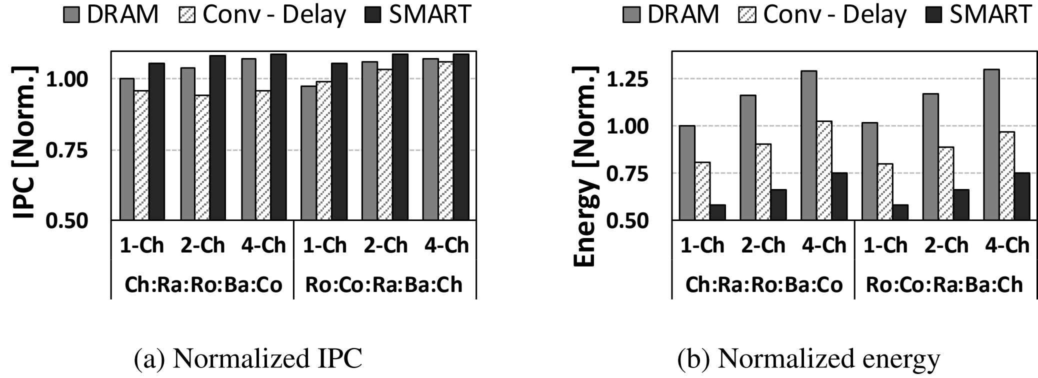
difference than SMART.

2.5.4 Sensitivity Analysis

Page size. Fig. 2.15 shows performance improvement and energy savings for various page sizes in conventional STT-MRAM. Overall, large pages improve performance while small pages save energy. The 256B page size slightly outperforms DRAM while saving less than 10% of energy. However, 256B (N=8) is not a practical page size considering the large size and high power consumption of the SAs. In prior STT-MRAM chip demonstrations, N ranges from 16 to 128 [27, 33, 37, 40, 48]. N=16 was conservatively selected for the baseline STT-MRAM.

Address mapping and channels. The results for two mapping schemes with 1$\sim$4 channels are shown in Fig. 2.16. DRAM is sensitive to the number of channels and performs the worst with a single channel because it is completely blocked during refresh. Generally, more channels increase the total bandwidth, but they also increase the number of memory chips and energy.

Conventional STT-MRAM is sensitive to the address mapping scheme. It performs better under Ro:Co:Ra:Ba:Ch because of its small pages. In general, the page size and row buffer locality are important to the Ch:Ra:Ro:Ba:Co mapping and the number of banks,



(a) Normalized IPC

(b) Normalized energy

Figure 2.16: Normalized IPC and energy to DRAM with various configurations (average of all mix workloads).

ranks, and channels plays a primary role under the Ro:Co:Ra:Ba:Ch mapping.

In contrast, SMART is less sensitive to address mapping and channels, because it has large pages, short row-miss latency, and no `ACT` to `ACT` constraint. In addition, there is no significant difference in energy savings according to the number of channels, because SMART reduces both dynamic and static energies.

2.6 Related Work

Kultursay *et al.* evaluate STT-MRAM as a main memory with optimizations such as partial write and write bypass [25]. They show comparable performance with DRAM and a 60% reduction in memory dynamic energy. Wang *et al*investigate the design challenges of shared SAs such as small pages and pin compatibility [42]. With memory-architectural study, they propose three optimizations, comboAS, DynLat, and EarlyPA. Although these studies solve the compatibility problem and compensate for the reduced performance, the root cause of small pages and low chip yield remains unsolved.

LPDDR2-SX was designed for DRAM and its counterpart LPDDR2-NVM was designed for non-volatile devices with long write latency and large read/write circuits such as PCM [62, 63]. LPDDR2-NVM introduces a new command to deliver column selection rather than using additional pins. Although it can be a good candidate for STT-MRAM,

its inherent performance is worse than LPDDR2-SX because of its three-phase addressing and software managed indirect write. Recently, 4Gb STT-MRAM has been demonstrated with LPDDR2-SX but not LPDDR2-NVM [27, 28].

SALP-1 [59] and EarlyPA [42] are similar to our skipping precharge in terms of alleviating `PRE` overhead. Unlike our technique, however, SALP-1 can only overlap `PRE` and next `ACT` when their sub-arrays are different. In EarlyPA, the precharge operation is automatically performed immediately after the sensing operation. Although this technique can efficiently hide the precharging time when the following command is `RD`, WL must be reactivated when the following command is `WR`. In contrast, our technique can hide `PRE` latency for both `RD` and `WR`.

2.7 Summary

We proposed SMART, co-designing STT-MRAM architecture and its management policy. By performing the sensing operation after receiving a `RD` command instead of a `ACT` command, SMART takes several advantages including larger pages, fewer sense amps, lower activation/sensing power, shorter latency, fewer address pins and more efficient column repair scheme over conventional STT-MRAM. With these benefits, SMART not only reduces energy but also improves performance compared to both DRAM and conventional STT-MRAM. In addition to the improvements in energy consumption and performance, SMART saves area from conventional STT-MRAM.

CHAPTER 3

Improving Load Balancing for Memory Channels

The performance needs of memory systems caused by growing volumes of data from emerging applications, such as machine learning and big data analytics, has continued to increase. As a result, HBM has been introduced in GPUs and throughput oriented processors. HBM is a stack of multiple DRAM devices across a number of memory channels. Although HBM provides a large number of channels and high peak bandwidth, we observed that all channels are not evenly utilized and often only one or few channels are highly congested after applying the hashing technique to randomize the translated physical memory address.

To solve this issue, we propose a cost-effective technique to improve load balancing for HBM channels. In the proposed memory system, a memory request from a busy channel can be migrated to other non-busy channels and serviced in the other channels. Moreover, this request migration reduces stalls by memory controllers, because the depth of a memory request queue in a memory controller is effectively increased by the migration. The improved load balancing of memory channels shows a 10.1% increase in performance for GPGPU workloads.

3.1 Introduction

Graphic Processing Units (GPUs) have developed for 3D graphics, games, and animations, and evolved for general purpose high performance computing [83–85]. GPU's on-chip computing capability has been improved rapidly in the past two decades [86]. However, the scaling of off-chip memory bandwidth has not followed the increasing computing capability. Thus, memory bandwidth often becomes the bottleneck limiting application performance [87]. Traditionally, GDDR, which is throughput-optimized DDR and whose the latest generation is GDDR5, memories have been used for GPUs. However, GDDR5 has challenges in increasing memory bandwidth, because its interface is narrow (16 or 32 per chip) and fast (up to 7Gbps per pin). Although high date rate is good for high bandwidth, it can be achieved by consuming high power. In addition, the small number of I/Os provided from GDDR5 requires many memory chips to be accommodated in GPUs to achieve high bandwidth. As a result, the required power and area make GDDR5 prohibitive beyond 1 TB/s of memory bandwidth [88].

High Bandwidth Memory (HBM) has been developed to overcome limited bandwidth of GDDR5 under the given power budget and form factor [35, 89, 90]. HBM is an on-package stacked DRAM and provides high peak bandwidth (∼256 GB/s) through multiple (up to 8) and wide channels (128 I/Os per channel). For the power efficiency, data rate (ı.e, double of clock speed in DDR) and thus supply voltage are lowered in HBM, but the increased number of I/Os and channels results in higher peak bandwidth than that of GDDR5-based GPUs. In other words, the high peak bandwidth of HBM stems from a number of memory channels. Therefore, high bandwidth can be achieved in HBM when all HBM channels are utilized well. However, it is hard to evenly utilize all memory channels for all applications because each application has different memory access pattern. Moreover, substantial imbalance on memory channels still remains after applying XOR-based address mapping scheme, which randomizes the address mapping to avoid excessive contention on one or few memory channels/banks [91, 92].

To address this issue, we propose a cost-effective technique to improve load balancing for HBM channels. Our technique is conceptually similar with the work stealing technique used in multi-core scheduling, where an idle core steals a work item in a busy core and the load is balanced across multiple cores [93, 94]. Similarly, in our proposed HBM-based memory sub-system, a memory request in a busy channel is migrated to other non-busy channel and issued through that channel. Then, the migrated request is rerouted to its original memory device. However, in traditional GDDR5-based memory sub-systems, this simple load balancing technique is hard to apply because of mainly two reasons. First, memory controllers and their physical channels are placed on the different side of the host processor chip. Thus, the memory request migration requires global interconnection across whole chip. Second, in order to reroute the migrated requests, extra off-chip interconnections to connect all off-chip GDDR5 chips are needed. Considering the cost to implement extra internal and external interconnections, the load balancing on memory channels by migration and rerouting would be impractical in the traditional GDDR5-based system. However, unlike the GDDR5-based system, the HBM-based system has several advantages in implementing this load balancing technique. First, multiple memory controllers for one HBM are locally placed because they are connected to the same chip having multiple DRAM devices. Thus, the local interconnections can enable the memory request migration. Second, one HBM has 8 channel's DRAM devices and rerouting of the memory request can be performed inside of HBM. Because each DRAM die for a channel has the physical connection of all TSVs and this connection can be electrically controlled, a simple modification in HBM can implement the rerouting.

In our proposed memory system, if a channel is highly utilized whereas other channels are not, the memory request migration is triggered. Then, the migrated memory request is rerouted to its original DRAM device by controlling electrical connection of TSVs in HBM. Through this balancing technique, the imbalance on memory channels is reduced by 7% on average. Moreover, because this requests migration effectively increases the depth

of memory request queue in a memory controller by occupying other memory controller's queue, the stall by memory sub-system is reduced. These improved load balancing and queue depth, in turn, bring 10.1% of GPU performance improvement (up to 26%).

3.2 Background

3.2.1 Increasing Demand of Memory Capacity and Bandwidth

The increasing volume of data to be processed by machine learning and big data analytics demands data parallel architectures such as Single Instruction Multiple Data (SIMD) and Single Instruction Multiple Threads (SIMT) architectures [95, 96]. Especially, GPUs become the *de facto* standard and, in turn, the state-of-the-art servers in clouds and datacenters are equipped with GPUs to speed up general purpose computation (ı.e, General Purpose computing on Graphics Processing Units (GPGPU)) [85]. Because GPUs have been designed to improve the throughput of applications by spawning many threads simultaneously, the capacity and bandwidth of memory have played an essential role in building high performance applications. For example, in many programming models in GPGPU applications such as nearest neighbor classifiers, decision trees, and neural networks, the size of GPU memory often imposes limitations on the data of size, resulting in decreased performance by continually transferring data from the system's memory to the GPU's memory [97]. In addition, because many of GPGPU applications are memory intensive and sometimes exhibit irregularity in their memory access patterns, their performance is significantly affected by memory bandwidth [98].

3.2.2 High Bandwidth Memory

HBM and HBM-based systems. Memory bandwidth has been continuously increased to meet GPU performance growth. However, in traditional GDDR5-based systems, there are

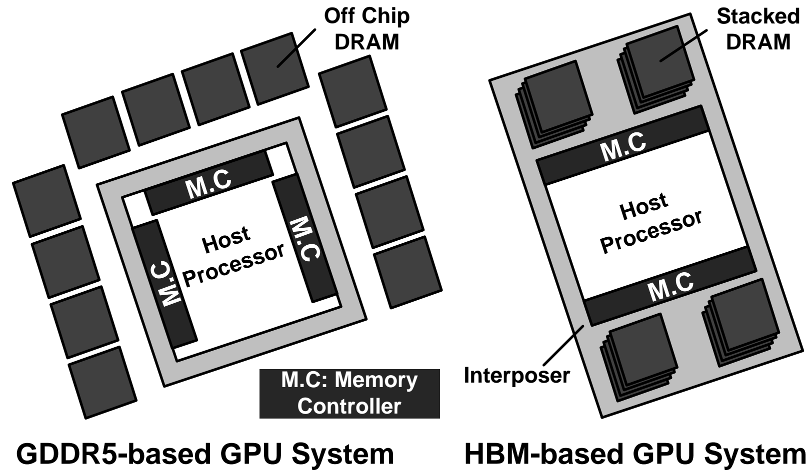


Figure 3.1: GPU systems with GDDR5 and HBM.

mainly two challenges in increasing memory bandwidth. First, the increased memory bandwidth brings a significant increase in the power budget for memory and this power budget is becoming prohibitive as the bandwidth scales beyond 1 TB/s [88]. Because GDRR5 is connected to a host processor through fast (up to 7Gbps per pin) and narrow (16 or 32 per chip) external I/O interface, its energy-per-bit presents high (∼14pJ/bit). Second, GDDR5 can limit form factors. As shown in Fig. 3.1(left), GDDR5 requires a large number of memory chips to reach high bandwidth because of its narrow channel. Also, to build a large memory system with a given density of GDDR5, more memory chips are needed. The large footprint by GDDR5 does not only affect form factors, but this also degrades the signal integrity on the memory interface because of long connection distance [99, 100].

HBM, which is an on-package stacked DRAM, has been introduced to overcome power and form factor challenges of GDDR5. Unlike GDDR5, HBM employs a slow (∼2Gbps per pin) and wide (128 per channel) channel and accordingly supply voltage becomes lowered (1.5V -> 1.2V)[1]. In addition, since HBM has multiple stacked DRAM dies and is connected to the host processor via silicon interposer within a package, this system can accommodate a large number of memory devices with a small space as shown in Fig. 3.1(right).

[1] In this study, we take HBM generation 2 (HBM2) as a baseline. However, we do not differentiate HBM and HBM2 in this study because main features of them, such as wide I/O, multi channels and 3D stacked DRAM dies, are almost same.

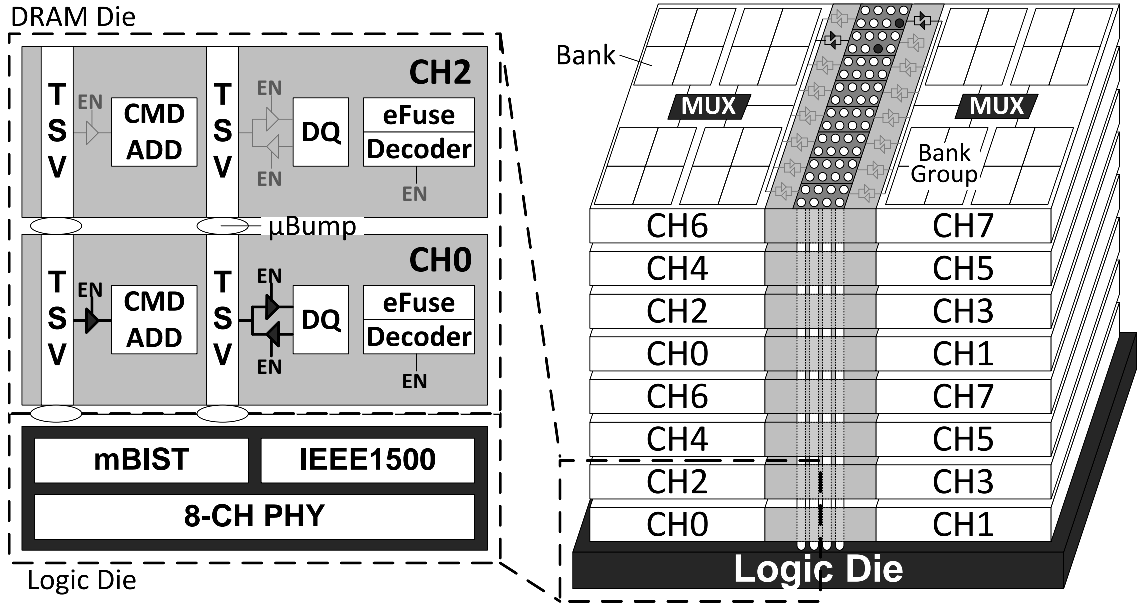
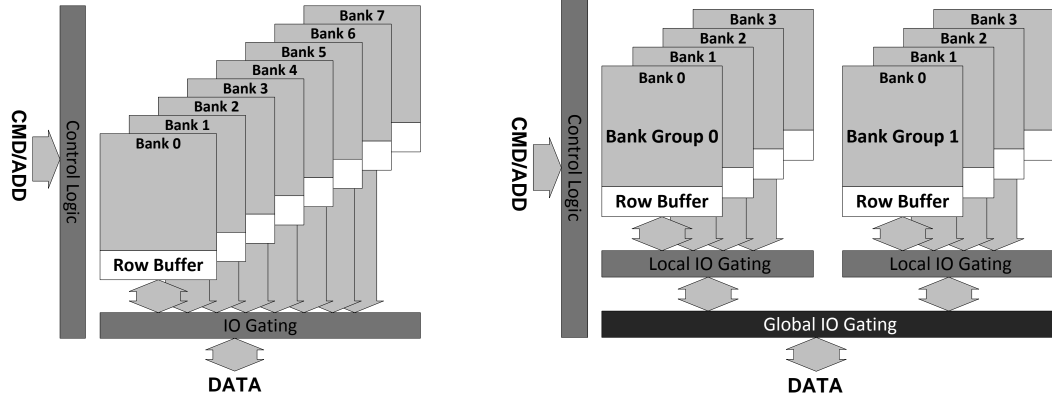


Figure 3.2: 3D structure of HBM and a simple example of TSV connection to DRAM dies.

TSV connections. Fig. 3.2 depicts the internal structure of HBM. An HBM is made with various capacity, the number of stacked layers and channel configurations [35]. In this study, the baseline HBM has 1Gb capacity and 2 half-channels per DRAM die and total 8 DRAM dies (total 8Gb). All DRAM dies are fabricated identically and thus all they are *physically* connected to TSVs for 8 channels. Then, a set of TSVs for certain channels can be *electrically* connected to one of the DRAM dies by using tri-state buffers with the decoder logic shown in the left of Fig. 3.2. During a manufacturing step, a Stack ID (SID) is programmed to the decoder to enable or disable the tri-state buffers by using electrical fuses (*e*fuses). We describe a example of these *physical* and *electrical* connections between TSVs and DRAM dies in Fig. 3.2, where the set of TSVs have *physically* connections to both DRAM dies but only the bottom DRAM die for `CH0` is *electrically* connected to the TSVs.

Bank group structure. The bank group feature, which is used in GDDR5 and DDR4, is adopted in HBM [5, 18, 35]. We describe the organization of a DRAM device with and without the bank group feature in Fig. 3.3. As shown in Fig. 3.3a, all banks are connected to one internal shared data bus. Traditionally, in order to bridge the gap between slow data



(a) Memory organization without bank-group (b) Memory organization with bank-group

Figure 3.3: Comparison between two memory organizations [5].

transfer speed on the shared bus and fast interface speed, data are transferred on the shared bus in parallel and then serialized out the interface with multiple clock cycles (*a.k.a n-prefetch*). In this structure, if the speed gap is increased, the prefetch length and accordingly burst length, which determines memory transaction and LLC line sizes, should be increased together to keep seamless burst read/write operations. Furthermore, bank-level parallelism in this structure is not improved well along with the number of banks because of limited scalability of the single shared bus. In order to avoid increasing prefetch length and improve parallelism, the bank group feature has been introduced as depicted in Fig. 3.3b. In the bank group structure, multiple banks groups (typically,4 or 8 groups, 4 by default in this study) have their own internal data bus and multiple banks (2 or 4 banks, 4 by default) in a bank group share one data bus. As the result of the separated data bus, multiple sets of data can be concurrently transferred between the interface and bank groups. However, different timing constraints, `tCCDS` and `tCCDL`, are applied when accessing banks in different bank groups and the same bank group, respectively. `tCCDL` is the minimum time between two read commands (or write commands) when accessing the same bank group and determined by the data transfer time on the shared data bus in the bank group. However, `tCCDS` is the minimum time between two read commands (or write commands) when accessing different bank groups and not determined by the data transfer time because two read accesses are

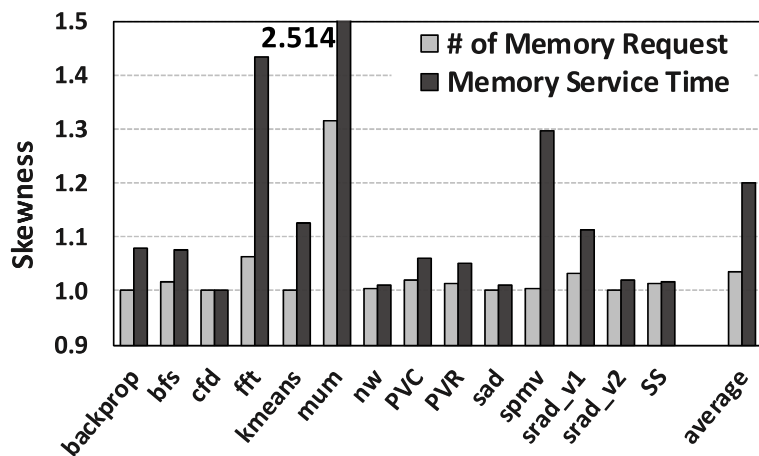


Figure 3.4: Channel Utilization.

served on different buses in different bank groups. Thus, the bank level parallelism in a bank group is still preserved, but the bank group level parallelism is a higher degree of parallelism.

3.3 Challenges in Many Channel Memory Systems

3.3.1 Imbalanced Channel Utilization

In general, the memory address mapping scheme is designed considering both spatial locality and parallelism [101]. For example, consecutive cache line accesses are scheduled to the same row in the same bank to take advantage of shorter latency when row buffer hit. On the other hand, accessing blocks of cache line alternates between multiple banks and channels by exploiting bank- and channel-level parallelism. However, depending on workloads memory system can suffer from excessive contention on one or few certain banks and channels. To prevent this situation, a permutation-based mapping scheme (*i.e*, hashing), in which channel and bank selection is determined by XORing a subset of MSB-side bits, has been proposed [91, 92]. Although this technique partially randomizes memory accesses, it is hard to completely eliminate the imbalanced memory requests on all channels and banks.

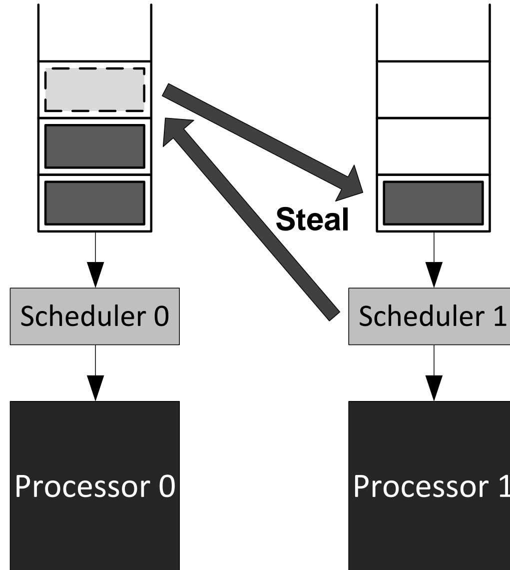


Figure 3.5: Simple diagram for work stealing.

Fig. 3.4 shows the skewness of total memory requests and service time across 8 channels of an HBM. The skewness is defined to the ratio of the minimum value to the maximum value. If the address mapping scheme is ideal and thus all channels receive the equal number of memory requests, the skewness of total memory requests becomes 1. Although the skewness of total memory requests is closed to 1 in many workloads due to XORing applied in the address mapping scheme, some workloads exhibit high skewness. Furthermore, this imbalance on the total number of memory requests is amplified on the service time, which is defined to the total time spent to serve all memory requests in a memory controller, as shown in Fig. 3.4. Because spatial and temporal locality in each channel can be different with the same of requests, they can make different scheduling scenario and result in non-equal memory service time in each channel.

The imbalanced memory requests and utilization across the memory channels can negatively affect overall performance by hindering exploiting full capability of all memory channels. Work stealing, which is well-known scheduling technique for multi-core systems, has been proposed to balance workloads and improve performance [93, 94]. We describe the simplified mechanism of the work stealing in Fig. 3.5. If a processor is idle

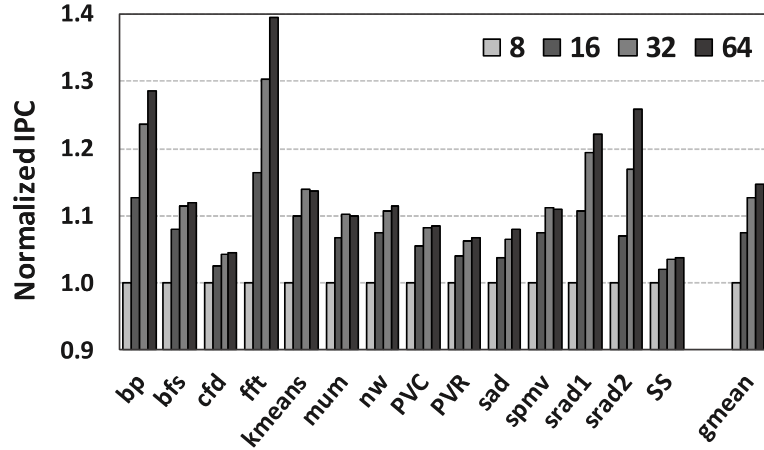


Figure 3.6: Performance according to queue depth.

(processor 1) is idle, it looks at the queue of another processor (processor 0) and steals its work if there are outstanding works. In this case, because all processors are identical, a work item can be executed in any processor. Therefore, the load balancing technique for memory channels like the work stealing can be considered to a good solution for imbalanced memory channels. However, the load balancing technique cannot be simply applied to the traditional GDDR5-based system, because each memory request (the work item in the work stealing) has its own memory address and it must be served in the preassigned memory device by the address. In other words, if a memory request is migrated to other channels and issued through the other channels, it must be rerouted and served in its initially assigned memory device.

3.3.2 Implementation Challenges of Memory Controllers

Having large request queues in the memory controller is generally beneficial to the performance because of mainly two reasons. First, the request queue is the buffer to mitigate the gap between fast input and slow service speeds of memory requests [102]. Thus, the queue depth determines the capability to hold the number of outstanding requests and this can significantly affect the performance when workloads are memory-intensive. Second,

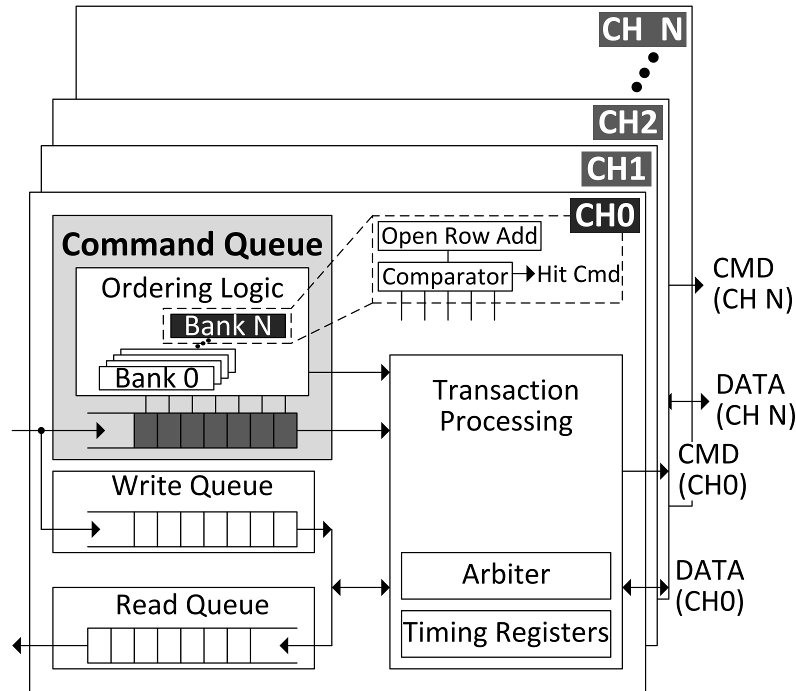


Figure 3.7: Schedulers of the memory controller in many channel memory systems [6].

there are more chances to make better scheduling decisions (i.e., shorter latency) in larger queues [103]. For instance, a First-Ready First-Come-First-Served (FR-FCFS [81, 104]) scheduler can make more row hits with a larger queue because the scheduler observes more memory requests and this increases the probability to find memory requests corresponding to the scheduling priority. However, there are fewer row hits with a smaller queue because of limited visibility to memory requests. Fig. 3.6 depicts the performance improvement according to the number of queue entries with various GPGPU applications. Based on workloads and their memory intensity, the sensitivity of performance improvement to the queue depth varies, but most workloads show higher performance with larger queues.

Unfortunately, in practical, it is hard to implement a sophisticated scheduling policy on a large queue. For example, in order to enable an FR-FCFS policy, the row address of all outstanding requests in the queue should be compared to the address of the already open row per bank every cycle [105, 106]. Such fully associative search demands Content Addressable Memories (CAMs). The design cost of CAM combining with the scheduling

logic super-linearly increases with the increase of the number of queue entries [103, 107]. Furthermore, as depicted in Fig. 3.7, each memory channel needs its own independent memory controller. Therefore, the area of memory controllers has a significant impact on total chip area in many channel memory systems.

3.4 Overview of the Proposed Design

In previous sections, we discussed why all memory channels are not evenly utilized and the request redistribution technique such as work stealing cannot be simply applied to memory systems. In addition, we observed performance improvement with large request queues in memory controllers, but it is hard to implement large queues with a FR-FCFS scheduling policy, because of the super-linearly increasing design cost as a result of the number of queue entries. With such observations, we propose a new memory system design to mitigate the imbalance of channel utilization and effectively increase the queue depth without increasing the actual queue size. In brief, our design allows memory requests to be inserted in, and issued from, any memory controller belonging to the same HBM. Then, the memory request issued through other channels is re-routed inside of the HBM. The bank-group feature enables us to concurrently serve multiple requests in the same memory device. There are three key observations which led to the proposed design; (1) multiple memory controllers for one HBM are placed locally, (2) in a HBM, all TSVs have physical connections to all DRAM dies and a set of TSVs constituting a channel can be electrically connected to any DRAM die by the decoder logic, and (3) multiple sets of data can be transferred concurrently inside of DRAM having bank group feature. In the remainder of this section, we first present the memory controller design and scheduling policy for our new design and then introduce the new HBM architecture.

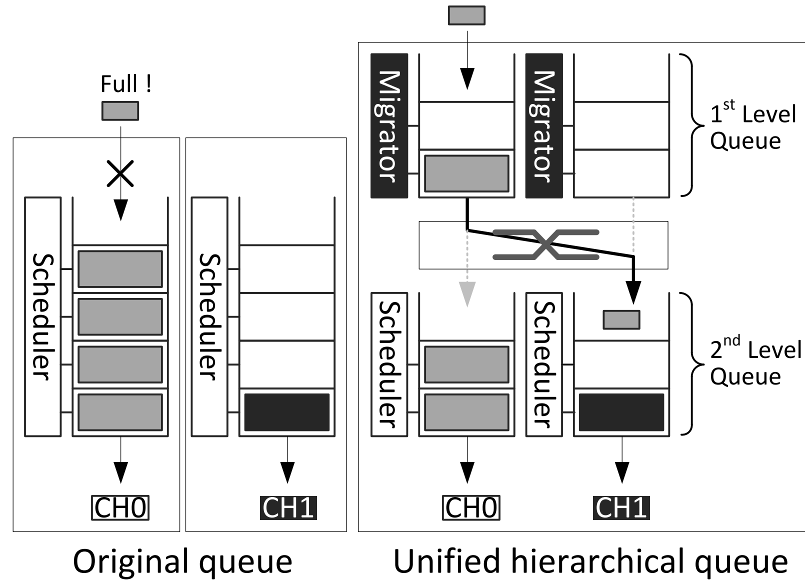


Figure 3.8: Hierarchical queue structure.

3.4.1 Re-architecting Memory Controllers

In general, each memory controller for each channel operates independently. In other words, each memory controller does not communicate with one another. There were studies to propose a technique to coordinate all memory controllers by connecting them to each other. Exchanging the scheduling status of each memory controller or globally applying a single scheduling priority can improve performance, because a single memory channel can be accessed by multiple threads and memory requests issued by a single thread can spread across multiple different channels. However, because memory controllers for different channels are often placed in opposite side of the chip as shown in Fig. 3.1, it is hard to implement the global interconnection between memory controllers in a traditional GDDR5-based system.

Unified Queue Structure. Unlike GDDR, where one chip provides only 32 I/Os and two chips compose one channel, one HBM provides 8 channels and accordingly the 8 memory controllers for the one HBM can be placed locally as shown in Fig. 3.1. Therefore, the interconnection between these memory controllers can be, also, implemented locally. With this observation, we propose a hierarchical queue structure as shown in Fig. 3.8. In our

hierarchical queue, one large queue is split into two smaller queues. Because of the super-linear relation between the size of a queue and area, we can save area by dividing the large queue. The saved area is used to implement crossbars. (Detail area analysis will be discussed in Sec. 3.5.3.)

In the proposed memory system, a memory request in a channel can be migrated to other channels having room to accept the memory request through the crossbar. In example of Fig. 3.8, each channel has a 4-entry request queue and channel 0 (`CH0`) is already full. In this case, the upper level (*e.g.*, last level cache) of the memory controller (*e.g*, last level cache) cannot issue a memory request to `CH0` and thus it is stalled, because there is no entry to accept the memory request in `CH0` as shown in Fig. 3.8(left). However, if a memory request in `CH0` is migrated to `CH1` and thus one empty entry is created in `CH0`, `CH0` can keep accepting memory requests without incurring stalls in its upper level (right in Fig. 3.8). Therefore, this technique can effectively increase queue depth and accordingly reduce the stall of the last level cache.

Channel Borrowing. In addition to the increased queue depth, we allow issuing the memory requests migrated from different channels. As a result, the migration reduces overall queuing delay because the memory requests migrated from a busy channel and issued through idle (or less busy) channels do not experience long waiting time in the queue. Note that the memory request issued through a different channel from its original channel must be re-routed to its original channel's DRAM device. Also, a DRAM device should equip the capability to handle more than two memory requests at the same time because multiple requests can be sent to the same DRAM device through different channels. To address these issues, we exploit the facts that all DRAM dies (all channels) have physical connections to all TSVs and the bank-group structure is capable of serving multiple requests concurrently. The cost-effective implementation inside of HBM will be discussed in the next section.

There are several challenges in scheduling the memory requests migrated from other channels. First, each memory controller must consider the timing constraints and bank

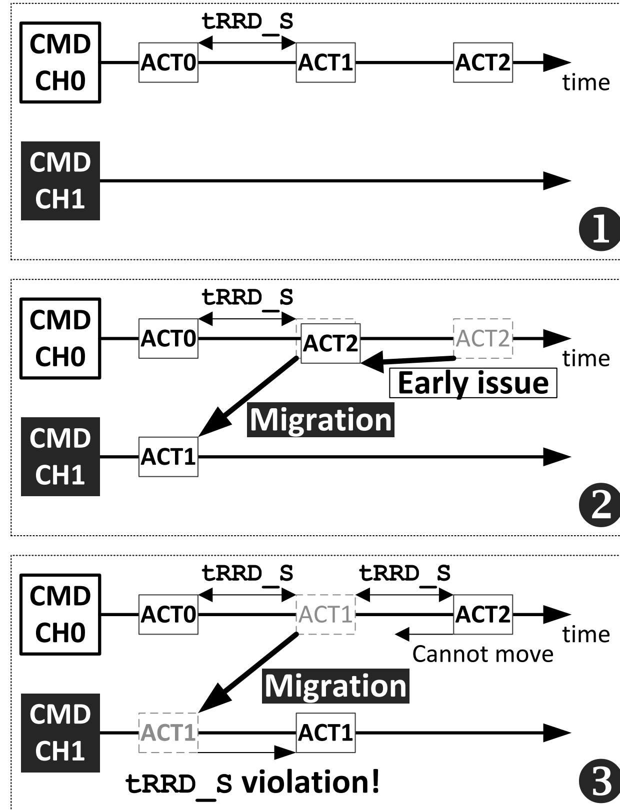


Figure 3.9: Limited scheduling.

status of all other channels to avoid command/data collision and timing violations. DRAM has various timing constraints which must be considered in scheduling memory requests in order to guarantee correct memory operations inside of DRAM (*e.g*, `tRCD`[2]), provide the power recovery time after high power consumption (*e.g*, `tRRDS`[3]) and avoid data collision on the memory bus (*e.g*, `tCCDS`). Hence, the scheduler in a memory controller has to abides by all timing constraints for all other channels as well as that for its channel when issuing memory requests. In Fig. 3.9, for example, `CH0` has three memory requests requiring a activation command (`ACT`) and `CH1` has no request to issue (❶). In this case, if only `tRRDS` for each channel is considered, the second `ACT` command (`ACT1`) in `CH0` can be migrated to `CH1` and can be issued through `CH1`'s memory bus and `ACT2` can be issued earlier in `CH0` (❷). Although `ACT0` and `ACT1` can be issued through different channels,

[2]minimum time between activation and read/write commands
[3]minimum time between two activation commands for different banks in different bank group

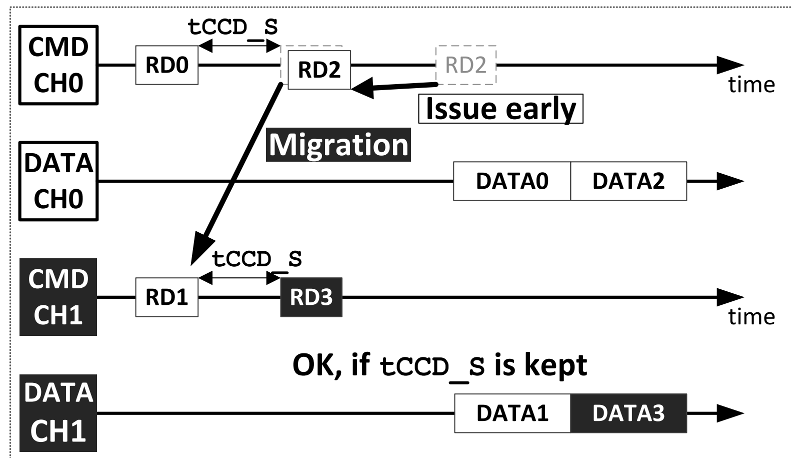


Figure 3.10: Avoiding timing constraint.

they will meet in the same DRAM device and make a `tRRDS` violation. In order to avoid the violation, `ACT1` in `CH1` must be issued after `tRRDS` is elapsed from `ACT0` issued in `CH0`. In addition to the delayed `ACT1`, `ACT2` in `CH0` must consider when `ACT1` is issued in `CH1` and cannot be issued earlier as shown in Fig. 3.9(❸). Therefore, this scheduling example does not have any benefit. Furthermore, this unuseful scheduling is even possible only when each memory controller considers the timing constraints of all channels and bank status of all other memory controllers.

Second, it is hard to determine the priority of the memory requests with the mixed memory requests having different channel addresses. As discussed in Sec. 3.3.2, a FR-FCFS scheduler compares the address of all outstanding requests to already open row addresses of all banks. Because a bank address of the memory request migrated from other channels should be considered by another independent bank regardless of the same bank address (*e.g.*, `BANK0-CH0` and `BANK0-CH1`), this request migration can effectively increase the number of banks to be considered for the scheduling. Thus, the scheduling complexity and thus the design complexity increase by the increased number of banks.

Considering the two challenges described above, the scheduling memory requests with the requests migrated from other channels is practically impossible. In order to overcome these challenges, we only migrate the memory requests which meet the predefined condi-

Rule 1: Migration conditions at the 1st level

1. **Full of their 2nd level queue—**Only when the 2nd level queue is full, requests are migrated to other channels. Normally, requests are served in their original channel.
2. **Room of other 2nd level queue—**Only when the 2nd level queue has enough room, where more than half of entries are not occupied, requests are migrated to this queue.
3. **Different bank group—**Requests having different bank group address from that of the outstanding requests in the 2nd level in the same memory controller are migrated to avoid collision on the internal memory I/O for the same bank group.
4. **Column command—**Requests having no need to issue row commands are migrated to avoid timing violation inside of DRAM.

tions. The first condition is that the memory request has the different bank group address from that of all outstanding memory requests in the second level queue. Because we exploit bank group level parallelism inside of DRAM, the memory requests having different bank group address can be served in DRAM at the same time. In other words, if the memory requests having the same bank group address are issued through the different channel at the same time, DRAM cannot accept all of them because there is only one shared internal I/O for a bank group inside of DRAM. Second, the memory request having currently open row's address is migrated to other channels. In other words, row commands (*i.e*, `ACT` and `PRE`) have to be issued in the original channel and column commands (*i.e*, `RD` and `WR`) can be issued in any channel. This second condition enables the avoidance of all timing violations related internal memory operations (*e.g*, `tRCD` and `tRRD`). Then, the migrated memory requested can be treated as a native memory request in a memory controller. As shown in Fig. 3.10, `RD1` migrated from `CH0` can be issued through `CH1` unless it violates `tCCDS` of `CH1`, which is `tCCD` for different bank group access and thus `RD2` in `CH0` can be issued earlier. Rule 1 summarizes all conditions for the migration.

Rule 2: Scheduling priorities at the 2nd level

1. **Migration—**Migrated requests which are always ready to issue are prioritized over native requests.
2. **Open row (FR-FCFS)—**Row-hit requests are prioritized over row-miss requests. This priority is only applied to native requests.
3. **Arrival time (FCFS)—**Older requests are prioritized over younger requests.

As we discussed earlier, the migration increases the number of banks to be considered in scheduling requests. However, in the proposed memory system, the memory requests going to the open row are only migrated. In other words, all migrated requests are ready to issue unless they violate `tCCDS`. Rule 2 describes the priorities for the scheduling decision at the second level of queues. Note that FR-FCFS requires the same number of comparators with the number of banks (typically, 16). However, to search the migrated requests only one small comparator is enough, because unlike the comparators for FR-FCFS which compare row address (15 bits for the baseline HBM) per bank, the comparator for the migrated requests only compares channel address (3 bits for the baseline HBM).

3.4.2 Re-architecting HBM

As we discussed in Sec. 3.2.2, all channels in a HBM have physical connections to all DRAM dies and a set of TSVs constituting a channel can be electrically connected to any DRAM die. In addition, the bank group structure enables DRAM to concurrently transfer multiple requests inside of DRAM because each bank group has an individual separated I/O. Motivated by these observations, we introduce alternative paths inside of HBM to serve more memory requests as shown in Fig. 3.11. In the original design, the DRAM die assigned for `CH7` is only electrically connected to a set of TSVs constituting `CH7` and a memory request coming from the TSVs of `CH7` is transferred to bank group 0 through the 4:1 muxes/demuxes. Because in the original design, only one set of TSVs is connected to

this DRAM die and only one memory request is issued to a channel at a time, one set of muxes/demuxes can relay the memory request to a bank group and only one bank group can receive a memory request at a time. However, in our proposed HBM, any DRAM die is electrically connected to any set of TSVs by the crossbars. Hence, a memory request coming from any channel can be relayed to any bank group through the 8:4 crossbars, consisting of sets of 8:1 muxes/demuxes and 4:1 muxes/demuxes as shown in Fig. 3.11(right). Based on the channel address of a request, the tri-state buffers and the first stage multiplexers (8:1 muxes) are controlled. Then, the bank group address is used for the second stage multiplexers (4:1 muxes). In Fig. 3.11(right), for example, two memory requests are sending to the same DRAM die of `CH7` through `CH0`'s and `CH7`'s TSVs, respectively. Although the two requests are issued from different memory controllers and transferred through different channel's buses and TSVs, they are relayed to the same DRAM die because of their same channel address. However, they have different bank addresses and are eventually arrive in different bank groups. Also, no collision of memory requests occurs in the crossbars (*i.e*, the requests coming from different TSVs, but going to the same bank group), because memory requests having different bank group addresses can be issued through different channels at the same time in the new HBM.

3.4.3 Overhead

To enable the migration of memory requests, we introduce extra circuits and storage. In this section, the overhead of our proposed memory system is discussed.

In memory controllers. At the first level of the request queue, in order to search the migration candidate, our design requires similar comparison logic with the FR-FCFS scheduler. In addition, the table keeping track of the bank group status is required. Second, the 8:8 crossbars are introduced to connect memory controllers for all channels for one HBM. Last, in order to differentiate the channel address of a request, 3 bits are added to each entry of the second level queue. However, because the queue depth of each level queue is half of

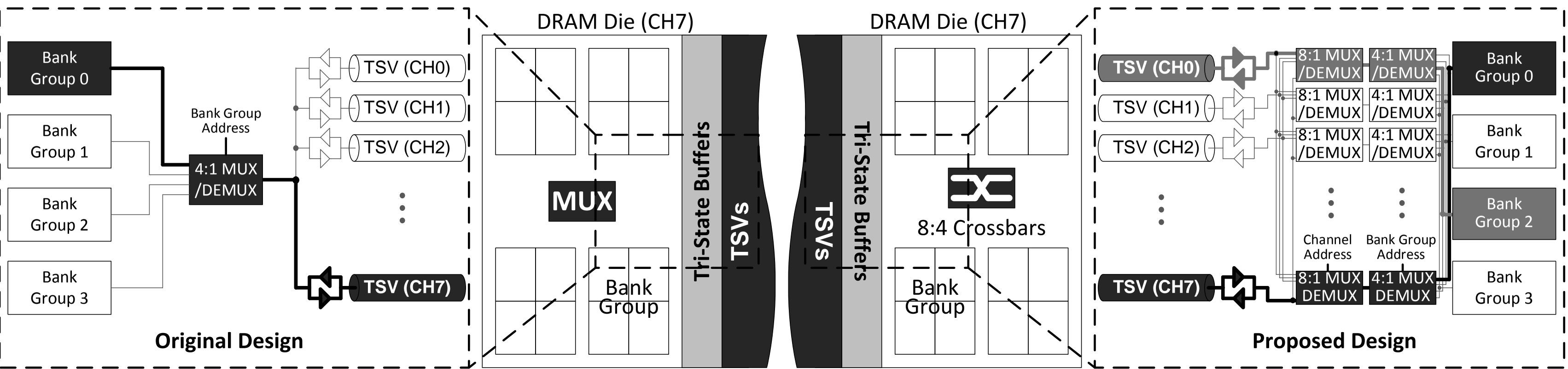


Figure 3.11: HBM with crossbars.

the baseline queue, we, actually, can save area in spite of the extra circuits and storage. The detail about the area will be discussed in Sec. 3.5.3 with various queue configurations.

In HBM. In order to reroute the migrated memory requests, we use the 8:4 crossbars and extra control signals. The estimated area increase using CACTI-3DD [108] and 20nm DRAM technology information [109] is 1.5% of a DRAM die.

3.5 Evaluation

3.5.1 Methodology

Table 3.1: Configured System

Component	Specification
Number of SM	15
Maximum Threads per SM	1536
L1 Data Cache per SM	16 KB
Number of Memory Channel	8
L2 Cache per Memory Channel	128 KB
Compute Core / Interconnect / Memory Clock	1000/1000/1000 MHz
DRAM Scheduling Policy	FR-FCFS
HBM Configuration per channel	8Gb, 128 I/Os, 2KB pages, 4 bank groups, 4 banks per bank group
HBM Timing Parameters (tCK)	tRC=47, tRCD=14, tRP=14, tRRDS=4, tRRDL=6, RL=14, WL=2, tCCDS=1, tCCDL=2, tRTPS=3, tRTPL=4, tWR=14

For our evaluation, we use GPGPU-Sim version 3.2.2 and implement our technique in its memory system [110]. The configured system for the evaluation is summarized in Table. 3.1. We model HBM based on [88] and present its key parameters in Table. 3.1. In order to evaluate our load balancing technique, we use several GPGPU benchmark suites such as MARS [111], Rodinia [112], Parboil [113] and mummerGPU (mum) provided in GPGPU-Sim. The workloads used in the evaluation are listed in Table. 3.2. We run all benchmarks for their full length to capture whole characteristics of them. For the baseline,

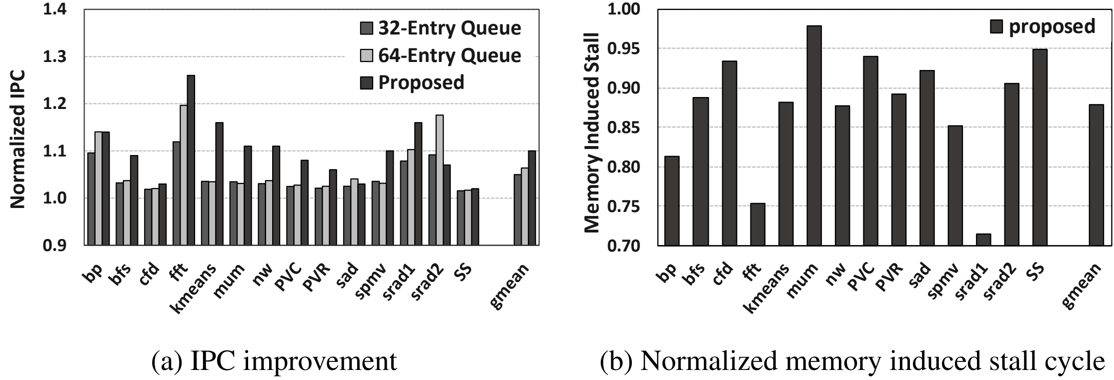
each memory controller has request queues of depth 16. In our memory system, the first level and second level queues have 8 entries, respectively. However, the second level queue can be shared by other channels, if memory requests meet the migration conditions.

Table 3.2: Workload list

Suite	Benchmark (abbreviation)
MARS	Page View Count (PVC), Page View Rank (PVR), Similarity Score (SS)
Parboil	Fast Fourier Transform (fft), Sum of Absolute Difference (sad), Sparse Matrix Dense Vector Multiplication (spmv)
Rodinia	Back Propagation (bp), Breath First Search (bfs), Computational Fluid Dynamics Solver (cfd), K-means Clustering (kmeans), Needleman-Wunsch Algorithm (nw), Speckle Reducing Anisotropic Diffusion version 1 (srad1), srad version 2 (srad2)

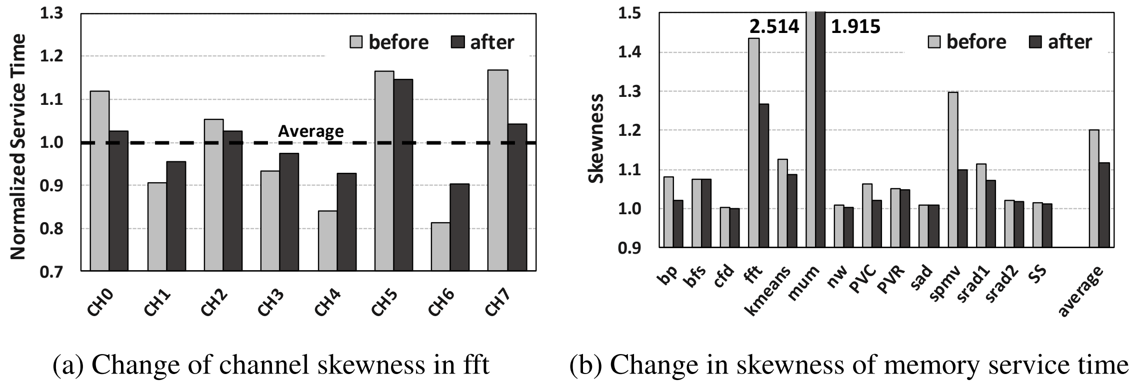
3.5.2 Performance Analysis

Fig. 3.12a plots the normalized instruction per cycle (IPC) to the baseline after applying our load balancing technique. The 32-entry queue and 64-entry queue are the same systems with the baseline except for the queue depth. As we discussed previous sections, having large queues is good for performance. As shown in Fig. 3.12a, our memory system outperforms the baseline and large queue systems. The improved IPC over the baseline is 10.1% on average and up to 26.0%. Because the request migration effectively increases the request queue depth, our memory system can hold more memory requests without incurring stall at the upper level. In *bp* and *srad1*, their performance is sensitive to queue depth and the increased queue depth in SMART mainly results in their performance improvement. However, in some applications (*e.g*, *bfs*, *mum* and *nw*), the queue depth does not have significant impact on the performance. In these applications, our memory system still brings performance improvement. This is because, in the proposed memory system, memory requests can be issued through different channels from their original channel, which reduces overall



(a) IPC improvement (b) Normalized memory induced stall cycle

Figure 3.12: Performance improvement after the migration.



(a) Change of channel skewness in fft (b) Change in skewness of memory service time

Figure 3.13: Change of imbalanced memory service time.

queuing delay by migrating blocked requests from the busy queue. Fig. 3.12b shows the normalized stall cycles of memory controllers, in which memory controllers cannot accept memory requests because there is no empty entry in their request queue. Our load balancing technique does not specifically prioritize certain critical requests (*e.g*, the requests determining the degree of memory divergence), but it reduces overall latency of memory requests. Thus, the reduced stall cycles in memory controllers are mostly reflected in the performance.

In Fig. 3.13, we present service time of each channel in *fft* and the skewness of the service time. In *fft*, the skewness of the number of memory requests was ∼1.06, which means there is only a 6% inequality in the number of requests between channels. However, the skewness of the memory service time, which is defined to be the active time of

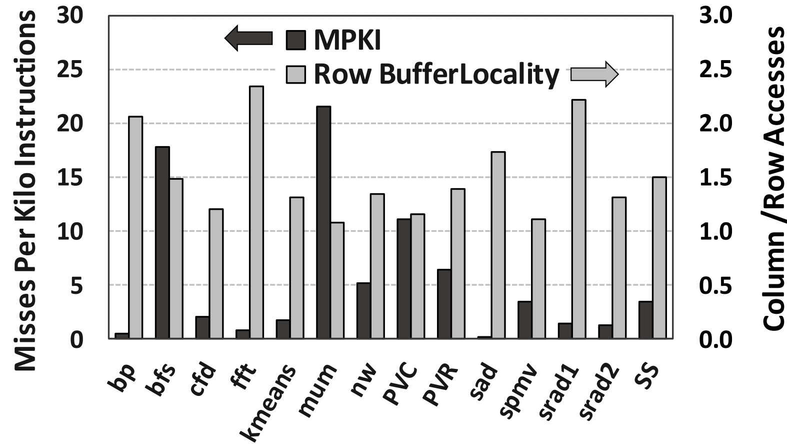


Figure 3.14: MPKI and locality.

memory controllers to serve the memory requests, was ∼1.44 (Fig. 3.4 in Sec. 3.3.1) and is much bigger than the skewness of the number of memory requests because of the difference in spatial and temporal localities between the channels. Because we migrate the memory requests from the busy channel to non-busy channel, this load balancing results in 12% reduction in the skewness of the service time. As shown in Fig. 3.13a, the busy channel becomes less busy and the non-busy channel becomes busier in our memory system. Fig. 3.13b shows the improved load balance in the proposed memory system. Overall, the skewness is reduced by 7% across all workloads, and there is a substantial reduction in *mum* (24%).

As we discussed in Sec. 3.4.1 (Rule 1), not all memory requests are migrated to other channels in our memory system. In order to obtain benefits from the migration, 1) the workloads should be memory intensive to generate enough congestion in the memory system, 2) the memory channels should be skewed in terms of their request service time and 3) the memory requests should have a certain degree of spatial locality to meet the migration conditions. We present the memory intensity of the workloads as misses-per-kilo-instructions (MPKI) and the spatial locality as the ratio of the number of column commands (`RD` and `WR`) to the number of row commands (`ACT`) in Fig. 3.14. A weak correlation between performance improvement and MPKI is observed, whereas the row buffer locality of the

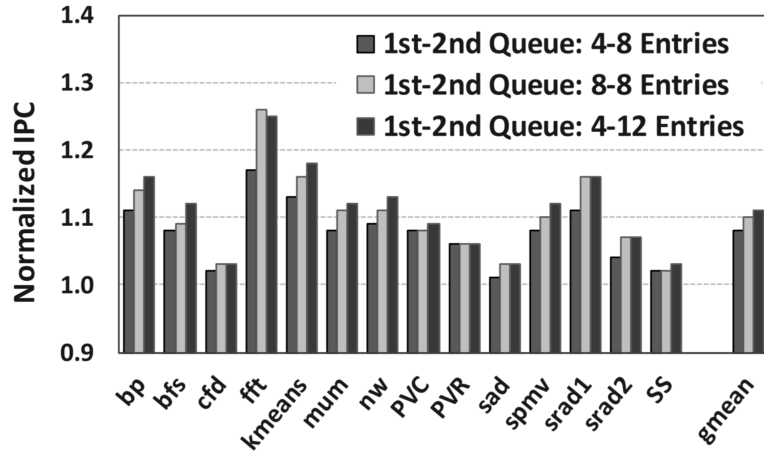


Figure 3.15: Performance according to different queue configuration.

workloads has a strong correlation with performance improvement.

In Fig. 3.15, we present performance improvement with various queue configurations. Because we effectively increase the queue depth, 4-8 configuration (4-entry for the first level queue and 8-entry for the second level queue), whose total number of queue entries (12) is smaller than the baseline (16), outperforms the baseline. Also, when the total number of queue entries is same, the 4-12 configuration shows slightly higher IPC than the 8-8 configuration. Because the migration can happen when memory controllers have more than half of empty entries at the second level queue, having a larger second level queue can permit more migrations and yield higher performance improvement than the smaller second level queue. The difference in performance between 8-8 and 4-12 configurations is 1-3%.

3.5.3 Area Overhead

In order to estimate the area of the memory controller, we developed Verilog models synthesized with a 45nm design library [114]. For this estimation, we only use standard cells. That is, all memory-components such as CAMs, buffers, and scheduling tables are modeled using flip-flop but not customized cells, and are therefore conservative. We present the

estimated area in Table. 3.3 Although we introduce crossbars and few extra logic, it is the reduced queue depth that mainly saves the area.

Table 3.3: Estimated area

Configuration	Area (normalized)
4-8	0.90
8-8	0.95
4-12	0.97

3.6 Related Work

Multi-Channel Memory Controllers. ATLAS is a scheduling technique proposed for fair scheduling across multiple memory channels [115]. ATLAS periodically orders threads based on the service they have attained from the memory controllers. After a long time quanta, information about the received service is exchanged between memory controllers and a central controller prioritize the threads that have attained the least service over others in the next epoch. Although ATLAS use a long time quanta to provide scalability, this approach is not practical for GPUs having thousands of threads.

Chatterjee *et al.* proposed a memory scheduling technique to manage memory latency divergence in GPUs [101]. In order to avoid inter-warp interference, they proposed forming batches of memory requests from a single warp called a warp-group. Also, their scheduling technique coordinates scheduling decisions across multiple memory channels with dedicated point-to-point interconnections between memory controllers.

Although these scheduling techniques can improve performance by coordinating scheduling decisions across multiple memory channels with given memory requests in a channel, there is no consideration about load balancing for memory channels.

Cost- and Complexity-Effective Memory Controllers. Staged Memory Scheduler (SMS) is a decentralized architecture for application-aware memory scheduling [103]. SMS decouples the memory controller's primary tasks and partitions them across simpler hardware

structures in a staged fashion. Because of the decentralized small request queues and simpler scheduling logic, SMS significantly saves area over FR-FCFS scheduler while improving performance. However, batch formation in SMS occurs an individual queue per thread. Thus, this scheduler is not suitable for GPUs having thousands of threads.

Yuan *et al*. addressed the high complexity of out-of-order scheduling such as FR-FCFS and proposed a complexity-effective solution for achieving the scheduling comparable to that of out-of-order scheduler [106] . Their key observation is that the row locality of the memory requests sent from the shader cores are much higher before they enter the interconnection network compared to when they arrive at memory controllers. By recovering the destroyed row locality in the interconnection network, their simple in-order memory controllers perform comparably with an out-of-order scheduler.

Similar to our design, they strove for reducing the implementation cost of memory controllers. However, both designs mainly focused on implementing an individual memory controller for a channel without consideration about the coordination between multiple channels.

3.7 Summary

The performance of memory systems often significantly affects overall system performance. HBM is optimized for high performance by providing a number of memory channels. Specifically, it is adapted to GPUs to meet their demand for high memory bandwidth. We observed that only one or a few memory channels are often highly utilized in GPGPU applications. This imbalance on memory channels hinders exploitation of the full bandwidth of an HBM. To overcome underutilized memory bandwidth, we propose a technique to improve load balancing for HBM channels. Our technique enables memory requests to migrate from a busy channel to other non-busy channels and service it in the other channels. In addition, the proposed technique effectively increases the depth of a request queue in a

memory controller, which results in the reduction of the stall cycles by memory controllers. Our load balancing technique mitigates the imbalance of the memory channel utilization and brings 10% of performance improvement for GPGPU applications.

CHAPTER 4

Conclusion

This dissertation focused on memory system design with emerging technologies to improve performance and energy efficiency. In this concluding chapter, we summarize the contributions of this dissertation and discuss their implications for future memory system design.

First, we started our study by investigating previous memory system designs with STT-MRAM. We found that most previous studies have not been fully exploited the advantages of STT-MRAM and thus there were lots of inefficiency in the previous designs. To address these issues, we proposed SMART, a new microarhictecture for an STT-MRAM main memory by fully exploiting the non-destructive nature of STT-MRAM cells. By moving the sensing operation from the row activation to the read operation, SMART realizes advantages in area, performance, and energy consumption. Our idea enables an area-efficient implementation of STT-MRAM main memory by reducing the number of SAs while keeping page size as large as DRAM, but without the overfetch problem. In addition, all operations regarding refresh, restoration, and precharge are removed in SMART, resulting in further improvement in both performance and energy consumption over DRAM. We demonstrated that an STT-MRAM main memory can be more energy-efficient and in some cases faster than DRAM.

In the second part of this dissertation, our study moved from main memory space to graphic memory space. Traditionally, GPUs have used performance-optimized memory

and recently HBM, which is 3D stacked DRAM. Although HBM provides high peak memory bandwidth through multiple and wide channels, we found that it is hard to fully utilize all of the bandwidth provided by HBM. Our observation was that all channels of an HBM are not evenly utilized and often a few channels are still highly congested even after applying the hashing technique to randomize the address of memory requests. To solve this issue, we proposed a cost-effective technique to improve load balancing for HBM channels. In the proposed memory system, a memory request from a busy channel can be migrated to other non-busy channels and serviced in the other channels. Moreover, the proposed technique effectively increases the depth of a request queue in a memory controller and this results in the reduction of stalls by memory controllers. Our load balancing technique mitigated the imbalance of the memory channel utilization and showed performance improvement for GPGPU applications.

In conclusion, the performance and energy efficiency of memory systems can be improved with emerging technologies, but they can be further improved if we fully exploit their hidden benefits. We believe that there are still undiscovered opportunities for emerging memory technologies.

BIBLIOGRAPHY

[1] Yoon, J. H., "3D NAND technology implications to enterprise storage applications," *Santa Clara, CA, USA: Flash Memory Summit*, 2015.

[2] Liu, J., Jaiyen, B., Veras, R., and Mutlu, O., "RAIDR: Retention-aware intelligent DRAM refresh," *ACM SIGARCH Computer Architecture News*, Vol. 40, IEEE Computer Society, 2012, pp. 1–12.

[3] McCalpin, J. D., "A survey of memory bandwidth and machine balance in current high performance computers," *IEEE TCCA Newsletter*, 1995.

[4] Micron, "8Gb DDR3L, MT41K1G8," 2015.

[5] MICRON, "DDR4 SDRAM," `https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_dram_2e0d.pdf`, 2014.

[6] Intel, "DRAM Controllers for System Designers," `https://www.altera.com/solutions/technology/system-design/articles/_2012/dram-controller-system-designer.html`, 2012.

[7] Von Neumann, J., "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing*, Vol. 15, No. 4, 1993, pp. 27–75.

[8] Hennessy, J. L. and Patterson, D. A., *Computer architecture: a quantitative approach*, Elsevier, 2011.

[9] Russo, J., "Stored program pay-per-play," April 8 1997, US Patent 5,619,247.

[10] Boncz, P. A., Manegold, S., Kersten, M. L., et al., "Database architecture optimized for the new bottleneck: Memory access," *VLDB*, Vol. 99, 1999, pp. 54–65.

[11] Mahapatra, N. R. and Venkatrao, B., "The processor-memory bottleneck: problems and solutions," *Crossroads*, Vol. 5, No. 3es, 1999, pp. 2.

[12] Mutlu, O., "Memory scaling: A systems architecture perspective," *Memory Workshop (IMW), 2013 5th IEEE International*, IEEE, 2013, pp. 21–25.

[13] Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., and Demmel, J., "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, Vol. 35, No. 3, 2009, pp. 178–194.

[14] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., McElroy, R., Paleczny, M., Peek, D., Saab, P., et al., "Scaling Memcache at Facebook." *nsdi*, Vol. 13, 2013, pp. 385–398.

[15] McDaniel, M. A. and Einstein, G. O., *Prospective memory: An overview and synthesis of an emerging field*, Sage Publications, 2007.

[16] Jacob, B., Ng, S., and Wang, D., *Memory systems: cache, DRAM, disk*, Morgan Kaufmann, 2010.

[17] Lindström, J., Raatikka, V., Ruuth, J., Soini, P., and Vakkila, K., "IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability." *IEEE Data Eng. Bull.*, Vol. 36, No. 2, 2013, pp. 14–20.

[18] JEDEC, "GRAPHICS DOUBLE DATA RATE (GDDR5) SGRAM STANDARD," https://www.jedec.org/system/files/docs/JESD212C.pdf, 2016.

[19] Zhang, T., Chen, K., Xu, C., Sun, G., Wang, T., and Xie, Y., "Half-DRAM: a High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation," *International Symposium on Computer Architecture (ISCA)*, 2014.

[20] Sudan, K., Chatterjee, N., Nellans, D., Awasthi, M., Balasubramonian, R., and Davis, A., "Micro-pages: increasing DRAM efficiency with locality-aware data placement," *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[21] Udipi, A. N., Muralimanohar, N., Chatterjee, N., Balasubramonian, R., Davis, A., and Jouppi, N. P., "Rethinking DRAM design and organization for energy-constrained multi-cores," *International Symposium on Computer Architecture (ISCA)*, 2010.

[22] Cooper-Balis, E. and Jacob, B., "Fine-grained activation for power reduction in DRAM," *IEEE Micro*, 2010.

[23] Asifuzzaman, K., Pavlovic, M., Radulovic, M., Zaragoza, D., Kwon, O., Ryoo, K.-C., and Radojković, P., "Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC," *International Symposium on Memory Systems (MEMSYS)*, 2016.

[24] Meza, J., Li, J., and Mutlu, O., "Evaluating row buffer locality in future non-volatile main memories," 2012.

[25] Kültürsay, E., Kandemir, M., Sivasubramaniam, A., and Mutlu, O., "Evaluating STT-RAM as an energy-efficient main memory alternative," *Performance Analysis of Systems and Software (ISPASS)*, 2013.

[26] Suresh, A., Cicotti, P., and Carrington, L., "Evaluation of emerging memory technologies for HPC, data intensive applications," *International Conference on Cluster Computing (CLUSTER)*, 2014.

[27] Rho, K., Tsuchida, K., Kim, D., Shirai, Y., Bae, J., Inaba, T., Noro, H., Moon, H., Chung, S., Sunouchi, K., et al., "23.5 A 4Gb LPDDR2 STT-MRAM with compact 9F2 1T1MTJ cell and hierarchical bitline architecture," *International Solid-State Circuits Conference (ISSCC)*, 2017.

[28] Chung, S.-W., Kishi, T., Park, J., Yoshikawa, M., Park, K., Nagase, T., Sunouchi, K., Kanaya, H., Kim, G., Noma, K., et al., "4Gbit density STT-MRAM using perpendicular MTJ realized with compact cell structure," *International Electron Devices Meeting (IEDM)*, 2016.

[29] Rizzo, N., Houssameddine, D., Janesky, J chand Whig, R., Mancoff, F., Schneider, M., DeHerrera, M., Sun, J., Nagel, K., Deshpande, S., et al., "A fully functional 64 Mb DDR3 ST-MRAM built on 90 nm CMOS technology," *IEEE Transactions on Magnetics*, 2013.

[30] Kim, C.-k., Kang, D.-s., Kim, H.-j., Park, C.-W., Dong-Hyun, S., Lee, Y.-S., Kang, S.-b., Oh, H.-R., Cha, S.-h., et al., "Magnetic random access memory," 2013, US Patent App. 13/768,858.

[31] Kim, H.-j., Kang, S.-K., Dong-Hyun, S., Kim, D.-M., and Lee, K.-C., "Magnetoresistive memory device including source line voltage generator," 2015, US Patent 9,036,406.

[32] Raychowdhury, A., Somasekhar, D., Karnik, T., and De, V., "Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances," *International Electron Devices Meeting (IEDM)*, 2009.

[33] Chung, S., Rho, K.-M., Kim, S.-D., Suh, H.-J., Kim, D.-J., Kim, H.-J., Lee, S.-H., Park, J.-H., Hwang, H.-M., Hwang, S.-M., et al., "Fully integrated 54nm STT-RAM with the smallest bit cell dimension for high density memory application," *International Electron Devices Meeting (IEDM)*, 2010.

[34] JEDEC, "Low Power Double Data Rate 3 (LPDDR3," `http://www.jedec.org/sites/default/files/docs/JESD209-3C.pdf`.

[35] JEDEC, "High Bandwidth Memory (HBM) DRAM," `https://www.jedec.org/sites/default/files/docs/JESD235A.pdf`, 2013.

[36] JEDEC, "Wide I/O 2 (WideIO2)," `https://www.jedec.org/system/files/docs/JESD229-2.pdf`, 2014.

[37] Kim, C., Kwon, K., Park, C., Jang, S., and Choi, J., "7.4 A covalent-bonded cross-coupled current-mode sense amplifier for STT-MRAM with 1t1mtj common source-line structure array," *International Solid-State Circuits Conference (ISSCC)*, 2015.

[38] Choi, Y., Song, I., Park, M.-H., Chung, H., Chang, S., Cho, B., Kim, J., Oh, Y., Kwon, D., Sunwoo, J., et al., "A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth," *International Solid-State Circuits Conference (ISSCC)*, 2012.

[39] Chung, H., Jeong, B. H., Min, B., Choi, Y., Cho, B.-H., Shin, J., Kim, J., Sunwoo, J., Park, J.-m., Wang, Q., et al., "A 58nm 1.8 v 1gb pram with 6.4 mb/s program bw," *International Solid-State Circuits Conference (ISSCC)*, 2011.

[40] Tsuchida, K., Inaba, T., Fujita, K., Ueda, Y., Shimizu, T., Asao, Y., Kajiyama, T., Iwayama, M., Sugiura, K., Ikegawa, S., et al., "A 64Mb MRAM with clamped-reference and adequate-reference schemes," *International Solid-State Circuits Conference (ISSCC)*, 2010.

[41] Lee, B. C., Ipek, E., Mutlu, O., and Burger, D., "Architecting phase change memory as a scalable dram alternative," *International Symposium on Computer Architecture (ISCA)*, 2009.

[42] Wang, J., Dong, X., and Xie, Y., "Enabling high-performance LPDDRx-compatible MRAM," *International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[43] Kim, D.-g. and Park, K.-t., "Semiconductor memory device with three-dimensional array and repair method thereof," Oct. 4 2011, US Patent 8,031,544.

[44] Yang, T.-C., Chih, Y.-D., and Liu, S.-H., "Redundancy circuits and operating methods thereof," Aug. 7 2012, US Patent 8,238,178.

[45] Worledge, D., Hu, G., Trouilloud, P., Abraham, D., Brown, S., Gaidis, M., Nowak, J., O'Sullivan, E., Robertazzi, R., Sun, J., et al., "Switching distributions and write reliability of perpendicular spin torque MRAM," *International Electron Devices Meeting (IEDM)*, 2010.

[46] Kitagawa, E., Fujita, S., Nomura, K., Noguchi, H., Abe, K., Ikegami, K., Daibou, T., Kato, Y., Kamata, C., Kashiwada, S., et al., "Impact of ultra low power and fast write operation of advanced perpendicular MTJ on power reduction for high-performance mobile CPU," *International Electron Devices Meeting (IEDM)*, 2012.

[47] Thomas, L., Jan, G., Zhu, J., Liu, H., Lee, Y.-J., Le, S., Tong, R.-Y., Pi, K., Wang, Y.-J., Shen, D., et al., "Perpendicular spin transfer torque magnetic random access memories with high spin torque efficiency and thermal stability for embedded applications," *Journal of Applied Physics*, 2014.

[48] Park, C., Kan, J., Ching, C., Ahn, J., Xue, L., Wang, R., Kontos, A., Liang, S., Bangar, M., Chen, H., et al., "Systematic optimization of 1 Gbit perpendicular magnetic tunnel junction arrays for 28 nm embedded STT-MRAM and beyond," *International Electron Devices Meeting (IEDM)*, 2015.

[49] Kan, J., Park, C., Ching, C., Ahn, J., Xue, L., Wang, R., Kontos, A., Liang, S., Bangar, M., Chen, H., et al., "Systematic validation of 2x nm diameter perpendicular MTJ arrays and MgO barrier for sub-10 nm embedded STT-MRAM with practically unlimited endurance," *International Electron Devices Meeting (IEDM)*, 2016.

[50] Song, B., Na, T., Kim, J., Kim, J. P., Kang, S. H., and Jung, S.-O., "Latch offset cancellation sense amplifier for deep submicrometer STT-RAM," *IEEE Transactions on Circuits and Systems I (TCAS I)*, 2015.

[51] Jefremow, M., Kern, T., Allers, W., Peters, C., Otterstedt, J., Bahlous, O., Hofmann, K., Allinger, R., Kassenetter, S., and Schmitt-Landsiedel, D., "Time-differential sense amplifier for sub-80mV bitline voltage embedded STT-MRAM in 40nm CMOS," *International Solid-State Circuits Conference (ISSCC)*, 2013.

[52] Na, T., Kim, J., Kim, J. P., Kang, S. H., and Jung, S.-O., "An offset-canceling triple-stage sensing circuit for deep submicrometer STT-RAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2014.

[53] Chang, M.-F., Shen, S.-J., Liu, C.-C., Wu, C.-W., Lin, Y.-F., King, Y.-C., Lin, C.-J., Liao, H.-J., Chih, Y.-D., and Yamauchi, H., "An offset-tolerant fast-random-read current-sampling-based sense amplifier for small-cell-current nonvolatile memory," *IEEE Journal of Solid-State Circuits (JSSC)*, 2013.

[54] Chang, M.-F., Sheu, S.-S., Lin, K.-F., Wu, C.-W., Kuo, C.-C., Chiu, P.-F., Yang, Y.-S., Chen, Y.-S., Lee, H.-Y., Lien, C.-H., et al., "A high-speed 7.2-ns read-write random access 4-mb embedded resistive ram (reram) macro using process-variation-tolerant current-mode read schemes," *IEEE Journal of Solid-State Circuits (JSSC)*, 2013.

[55] Cuppu, V., Jacob, B., Davis, B., and Mudge, T., "A performance comparison of contemporary DRAM architectures," *International Symposium on Computer Architecture (ISCA)*, 1999.

[56] JEDEC, "DDR3 SDRAM Specification," `www.jedec.org/sites/default/files/docs/JESD79-3E.pdf`, 2009.

[57] JEDEC, "High Bandwidth Memory (HBM) DRAM," `https://www.jedec.org/sites/default/files/docs/JESD235A.pdf`, 2013.

[58] Moon, Y., Cho, Y.-H., Lee, H.-B., Jeong, B.-H., Hyun, S.-H., Kim, B.-C., Jeong, I.-C., Seo, S.-Y., Shin, J.-H., Choi, S.-W., et al., "1.2 V 1.6 Gb/s 56nm 6F 2 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture," *International Solid-State Circuits Conference (ISSCC)*, 2009.

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

[60] Alam, S. M., Andre, T., and Gogl, D., "Memory controller and method for interleaving DRAM and MRAM accesses," 2016, US Patent 9,418,001.

[61] Yoo, J.-H., Kim, C. H., Lee, K. C., Kyung, K.-H., Yoo, S.-M., Lee, J. H., Son, M.-H., Han, J.-M., Kang, B.-M., Haq, E., et al., “A 32-bank 1 Gb DRAM with 1 GB/s bandwidth,” *International Solid-State Circuits Conference (ISSCC)*, 1996.

[62] Li, Z., Zhou, R., and Li, T., “Exploring high-performance and energy proportional interface for phase change memory systems,” *International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[63] JEDEC, “Low Power Double Data Rate 2 (LPDDR2),” http://www.jedec.org/sites/default/files/docs/JESD209-2B.pdf, 2009.

[64] Lim, K.-N., Jang, W.-J., Won, H.-S., Lee, K.-Y., Kim, H., Kim, D.-W., Cho, M.-H., Kim, S.-L., Kang, J.-H., Park, K.-W., et al., “A 1.2 V 23nm 6F 2 4Gb DDR3 SDRAM with local-bitline sense amplifier, hybrid LIO sense amplifier and dummy-less array architecture,” *International Solid-State Circuits Conference (ISSCC)*, 2012.

[65] Naji, O., Weis, C., Jung, M., Wehn, N., and Hansson, A., “A high-level DRAM timing, power and area exploration tool,” *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, IEEE, 2015.

[66] Min, K.-S. and Chung, J.-Y., “A fast pump-down V BB generator for sub-1.5-V DRAMs,” *IEEE Journal of Solid-State Circuits (JSSC)*, 2001.

[67] Lee, D. U., Lee, K. S., Lee, Y., Kim, K. W., Kang, J. H., Lee, J., and Chun, J. H., “Design considerations of HBM stacked DRAM and the memory architecture extension,” *Custom Integrated Circuits Conference (CICC)*, 2015.

[68] Horiguchi, M. and Itoh, K., *Nanoscale memory repair*, Springer Science & Business Media, 2011.

[69] Horiguchi, M., Etoh, J., Aoki, M., Itoh, K., and Matsumoto, T., “A flexible redundancy technique for high-density DRAMs,” *IEEE Journal of Solid-State Circuits (JSSC)*, 1991.

[70] Lin, C., Kang, S., Wang, Y., Lee, K., Zhu, X., Chen, W., Li, X., Hsu, W., Kao, Y., Liu, M., et al., “45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell,” *Electron Devices Meeting (IEDM), 2009 IEEE International*, IEEE, 2009, pp. 1–4.

[71] Oh, B.-C., Bae, J.-H., Fujita, K., and Shirai, Y., “Electronic device including semiconductor memory and operation method thereof,” 2014, US Patent 6,442,585.

[72] Nowak, J. J., Robertazzi, R. P., Sun, J. Z., Hu, G., Park, J.-H., Lee, J., Annunziata, A. J., Lauer, G. P., Kothandaraman, R., OSullivan, E. J., et al., “Dependence of voltage and size on write error rates in spin-transfer torque magnetic random-access memory,” *IEEE Magnetics Letters*, Vol. 7, 2016, pp. 1–4.

[73] Grezes, C., Lee, H., Lee, A., Wang, S., Ebrahimi, F., Li, X., Wong, K., Katine, J. A., Ocker, B., Langer, J., et al., “Write Error Rate and Read Disturbance in Electric-Field-Controlled Magnetic Random-Access Memory,” *IEEE Magnetics Letters*, Vol. 8, 2017, pp. 1–5.

[74] Saida, D., Kashiwada, S., Yakabe, M., Daibou, T., Fukumoto, M., Miwa, S., Suzuki, Y., Abe, K., Noguchi, H., Ito, J., et al., “1x- to 2x-nm perpendicular MTJ Switching at Sub-3-ns Pulses Below 100uA for High-Performance Embedded STT-MRAM for Sub-20-nm CMOS,” *IEEE Transactions on Electron Devices*, Vol. 64, No. 2, 2017, pp. 427–431.

[75] Dong, X., Xu, C., Xie, Y., and Jouppi, N. P., “NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012.

[76] ITRS, http://www.itrs2.net/2013-itrs.html, 2013.

[77] Park, J., Shin, D.-H., Cho, Y.-H., and Kwon, K.-W., “Inverted bit-line sense amplifier with offset-cancellation capability,” *Electronics Letters*, 2016.

[78] Oh, B., Abeyratne, N., Ahn, J., Dreslinski, R. G., and Mudge, T., “Enhancing DRAM Self-Refresh for Idle Power Reduction,” *International Symposium on Low Power Electronics and Design (ISLPED)*, 2016.

[79] Patel, A., Afram, F., Chen, S., and Ghose, K., “MARSS: a full system simulator for multicore x86 CPUs,” *Design Automation Conference (DAC)*, 2011.

[80] Rosenfeld, P., Cooper-Balis, E., and Jacob, B., “DRAMSim2: A cycle accurate memory system simulator,” *IEEE Computer Architecture Letters*, 2011.

[81] Rixner, S., Dally, W. J., Kapasi, U. J., Mattson, P., and Owens, J. D., “Memory access scheduling,” *International Symposium on Computer Architecture (ISCA)*, 2000.

[82] Henning, J. L., “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, 2006.

[83] Harris, M. and Luebke, D., “GPGPU: General-purpose computation on graphics hardware,” *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Courses: Los Angeles, California*, Vol. 2005, 2005.

[84] Kirk, D. et al., “NVIDIA CUDA software and GPU parallel computing architecture,” *ISMM*, Vol. 7, 2007, pp. 103–104.

[85] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J., “A survey of general-purpose computation on graphics hardware,” *Computer graphics forum*, Vol. 26, Wiley Online Library, 2007, pp. 80–113.

[86] Chen, J. Y., “GPU technology trends and future requirements,” *Electron Devices Meeting (IEDM), 2009 IEEE International*, IEEE, 2009, pp. 1–6.

[87] Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., and Glasco, D., "GPUs and the future of parallel computing," *IEEE Micro*, Vol. 31, No. 5, 2011, pp. 7–17.

[88] Chatterjee, N., OConnor, M., Lee, D., Johnson, D. R., Keckler, S. W., Rhu, M., and Dally, W. J., "Architecting an energy-efficient dram system for gpus," *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, IEEE, 2017, pp. 73–84.

[89] NVIDA, "Inside pascal: NVIDIA's newest computing platform," https://devblogs.nvidia.com/inside-pascal, 2016.

[90] AMD, "Inside pascal: NVIDIA's newest computing platform," https://www.amd.com/en/technologies/hbm, 2015.

[91] Vandierendonck, H. and De Bosschere, K., "XOR-based hash functions," *IEEE Transactions on Computers*, Vol. 54, No. 7, 2005, pp. 800–812.

[92] van den Braak, G.-J., Gomez-Luna, J., González-Linares, J. M., Corporaal, H., and Guil, N., "Configurable XOR hash functions for banked scratchpad memories in GPUs," *IEEE Transactions on Computers*, Vol. 65, No. 7, 2016, pp. 2045–2058.

[93] Blumofe, R. D. and Leiserson, C. E., "Scheduling multithreaded computations by work stealing," *Journal of the ACM (JACM)*, Vol. 46, No. 5, 1999, pp. 720–748.

[94] Mitzenmacher, M., "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 10, 2001, pp. 1094–1104.

[95] Gupta, P., Sharma, A., and Jindal, R., "Scalable machine-learning algorithms for big data analytics: a comprehensive review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 6, No. 6, 2016, pp. 194–214.

[96] Singh, D. and Reddy, C. K., "A survey on platforms for big data analytics," *Journal of Big Data*, Vol. 2, No. 1, 2015, pp. 8.

[97] Cano, A., "A survey on graphic processing unit computing for large-scale data mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 8, No. 1, 2018.

[98] Burtscher, M., Nasre, R., and Pingali, K., "A quantitative study of irregular programs on GPUs," *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, IEEE, 2012, pp. 141–151.

[99] Ikeda, H. and Inukai, H., "High-speed DRAM architecture development," *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 5, 1999, pp. 685–692.

[100] Loi, I. and Benini, L., "An efficient distributed memory interface for many-core platform with 3D stacked DRAM," *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2010, pp. 99–104.

[101] Chatterjee, N., O'Connor, M., Loh, G. H., Jayasena, N., and Balasubramonian, R., "Managing DRAM latency divergence in irregular GPGPU applications," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, 2014, pp. 128–139.

[102] Little, J. D. and Graves, S. C., "Little's law," *Building intuition*, Springer, 2008, pp. 81–100.

[103] Ausavarungnirun, R., Chang, K. K.-W., Subramanian, L., Loh, G. H., and Mutlu, O., "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, IEEE, 2012, pp. 416–427.

[104] Zuravleff, W. K. and Robinson, T., "Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order," May 13 1997, US Patent 5,630,096.

[105] Rotithor, H. G., Osborne, R. B., and Aboulenein, N., "Method and apparatus for out of order memory scheduling," Oct. 24 2006, US Patent 7,127,574.

[106] Yuan, G. L., Bakhoda, A., and Aamodt, T. M., "Complexity effective memory access scheduling for many-core accelerator architectures," *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2009, pp. 34–44.

[107] Palacharla, S., Jouppi, N. P., and Smith, J. E., *Complexity-effective superscalar processors*, Vol. 25, ACM, 1997.

[108] Chen, K., Li, S., Muralimanohar, N., Ahn, J. H., Brockman, J. B., and Jouppi, N. P., "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, IEEE, 2012, pp. 33–38.

[109] Semiconductor, S., "Research collaboration communications," 2016.

[110] Bakhoda, A., Yuan, G. L., Fung, W. W., Wong, H., and Aamodt, T. M., "Analyzing CUDA workloads using a detailed GPU simulator," *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 163–174.

[111] He, B., Fang, W., Luo, Q., Govindaraju, N. K., and Wang, T., "Mars: a MapReduce framework on graphics processors," *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, 2008, pp. 260–269.

[112] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K., "Rodinia: A benchmark suite for heterogeneous computing," *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Ieee, 2009, pp. 44–54.

[113] Stratton, J. A., Rodrigues, C., Sung, I.-J., Obeid, N., Chang, L.-W., Anssari, N., Liu, G. D., and Hwu, W.-m. W., "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, Vol. 127, 2012.

[114] University, O. S., "FreePDK: Unleashing VLSI to the Masses," https://vlsiarch.ecen.okstate.edu/flows/, 2017.

[115] Kim, Y., Han, D., Mutlu, O., and Harchol-Balter, M., "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, IEEE, 2010, pp. 1–12.